Milí řešitelé!

Pomalu přichází zima a abyste měli o dlouhých večerech co dělat, máme pro Vás druhou sérii našeho semináře. Neobsahuje ještě Vaše opravená řešení první série, to abyste na nové příklady nemuseli čekat moc dlouho. Nicméně opravená řešení Vám (spolu se zadáním třetí série) přijdou už brzy, dříve, než budete muset tuto druhou sérii odeslat.

Pokud chcete posílat svá řešení elektronicky, držte se instrukcí, které můžete najít na *http://ksp.mff.cuni.cz/submit/*. Naši počítačoví skřítci předem děkují.

Termín odeslání Vašich řešení druhé série jest stanoven na 10. prosince 2007 a naše adresa je stále stejná: **Korespondenční seminář z programování**
KSVI MFF UK
Malostranské náměstí 25
118 00 Praha 1

Aktuální informace o KSP můžete nalézt na stránkách *http://ksp.mff.cuni.cz/*, diskutovat můžete na fóru *http://ksp.mff.cuni.cz/forum/* a záludné dotazy organizátorům lze zasílat e-mailem na adresu *ksp@mff.cuni.cz*.

Druhá série dvacátého ročníku KSP

Mág popošel na příď lodi a zahleděl se do temnoty. A doopravdy, na druhém břehu se něco pohybovalo. A hned na několika místech. To budou zase ti temní elfové, pomyslel si mág a ledabyle naznačil Vildovi, aby si lehl na podlahu. Vilda okamžitě uposlechl a sotva se sehnul, prosvištělo mu nad hlavou několik šípů, které s tlumeným žbluňknutím zmizely v řece.

„To jsou celí oni," postěžoval si mág. „Nejdřív střílí, a pak teprve kladou otázky."

„No prosím, já vám říkal, abychom zůstali doma. Ale mě tady nikdo neposlouchá," přisadil si Felix a naježil srst.

„Kdo se opovažuje útočit na temného mága – pána tohoto hvozdu?!" zaburácel temný mág a jeho hlas se nesl na všechny strany s údernou silou vichřice. Chvíli počkal, než se vzduch opět uklidnil, a pak se otočil k posádce lodi. Vilda se třásl na zemi, Kiri visel na kormidle v poloze „netopýr" a Felixova srst vypadala, jako by právě proběhl silným elektro-magickým polem.

„M-mohl bys nás příště laskavě varovat, než použiješ svůj Hlas?" ozvalo se ze středu chlupaté koule, která byla ještě před chvílí kocourem.

„Snad se zas tak moc nestalo," ohradil se mág. „Alespoň nás teď elfové nebudou obtěžovat."

Dopluli ke břehu. Vilda vyskočil a uvázal loď k velkému kameni. Byl by použil strom, ale v temném hvozdu si jeden nemůže být jistý, kdy si takový strom vzpomene, že se půjde projít. I ostatní mezi tím vystoupili a poslechli si další variaci na ódu „já chci domů" v kočičím podání. Vilda posbíral vybavení a vydali se na cestu. Ušli sotva pár kroků, když se odnikud vynořila skupinka temných elfů a v mžiku je obklíčila.

Ach ne, už zase? Být temným mágem je těžší, než byste mysleli. Nejtěžší je umět rozhodnout, kdy magii použít, a kdy ne. A tohle bylo přesně jedno z těch rozhodnutí. Samozřejmě by nebyl problém proměnit všechny přítomné elfy ve slintající idioty, ale to by znamenalo komplikace. Ostatní elfové by to nepřešli bez povšimnutí. Lezli by mu do věže a on by se příšerně nudil při nekonečných rozhovorech s elfími delegacemi. Nebo by to rozpoutalo válku, a to by mu tak ještě na stará kolena chybělo. Raději si poslechne, co mu chtějí.

Elfové nepromluvili ani slovo. Jen taktně – tedy s napjatými dlouhými luky – naznačili, aby je následovali. Prodírali se hvozdem dobrou hodinu, než přišli do pravé temnoelfské vesnice. Běžný pozorovatel by ji možná ani nerozeznal od jiné části hvozdu, ale zkušené oko rychle odhalilo výdutě ve stromech a umně ukryté provazové žebříky. Naproti se k nim svižnou chůzí blížil drobný postarší elf v honosném rouchu, které již na první pohled budilo dojem nadřazenosti.

„Zdravím vás," zašvitořil přátelsky a ostatním elfům s luky naznačil, že se můžou ztratit. „Omlouvám se za komplikace, ale vypadá to, že došlo k menšímu nedorozumění. Náš bývalý šaman, kterému se mimochodem dnes ráno přihodila malá – ehm – nehoda, vydal jisté nesmyslné příkazy..." pokračoval elf s potměšilým úsměvem na rtech. „Zkrátka a dobře jste se sem dostali nedopatřením a já se vám jménem našeho kmene velice omlouvám. Budete-li mít zájem, můžete strávit noc v naší vesnici. Také jste zváni na slavnostní večerní rituál, při kterém bude zvolen nový šaman."

Mágova družina byla pěkně unavená, a tak velkorysou nabídku přijala. Rituál byl velice působivý a mága nesmírně zajímalo, jak to ti elfové vlastně dělají. Bohužel se žádný z nich nechtěl o prastaré tajemství jejich kmene podělit, a tak si mág řekl, že na to přijde sám...

20-2-1 Volba šamana — 7 bodů

Volba šamana je delikátní záležitost. Jedná se o prastarý rituál, při kterém je nejstarší elf zvolen šamanem. Před začátkem si všichni stoupnou do kruhu tak, že každý elf vidí pouze své dva sousedy a nikoho jiného. Takže např. ani nevědí, kolik celkem jich v kruhu je.

Začátek rituálu je oznámen úderem do speciálního bubnu z hroší kůže a bubnování pokračuje po celou dobu rituálu. Mezi dvěma po sobě jdoucími údery si může každý elf vyměnit krátké zprávy se svým kolegou vlevo i vpravo (tzn. může každému říct nějakou zprávu a také si jeho zprávu vyslechnout). Každý elf si může pochopitelně také něco pamatovat.

Všichni elfové musí ve stejném okamžiku vědět, že rituál již skončil. V okamžiku, kdy skončí, musí všichni vědět, kdo je novým šamanem (včetně šamana samotného).

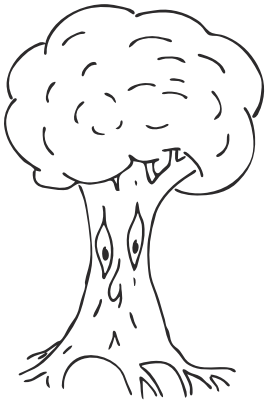
Aby tento rituál fungoval vždy, musí mít všichni elfové stejné instrukce (stejný algoritmus). Navíc nelze předpokládat, že by o sobě navzájem cokoliv věděli – tj. pro účely rituálu zná na počátku každý elf pouze své jméno a svůj věk (o ostatních neví nic). Pro jednoduchost předpokládejte, že žádní dva elfové nejsou stejně staří.

Navrhněte algoritmus, který musí každý elf provádět, aby rituál fungoval, tedy aby byl vždy zvolen nejstarší elf.

Ráno se temná družinka nasnídala a vyrazila směrem z temného hvozdu. Cesta ubíhala pomalu a ani sám mág si nebyl tak docela jistý, jestli jdou dobře. Stromy se občas přesazují, jak se jim zlíbí, takže není vůbec jednoduché se v temném hvozdu orientovat. Několik hodin bloudili a chodili stále dokola.

„Přísahal bych, že tenhle strom už jsem viděl," zamumlal si mág pod vousy.

„Ale však já už vás také několikrát viděl," přitakal strom.



„Takhle byste se z hvozdu nikdy nedostali. Běžte pořád tímhle směrem, až dorazíte na louku. Tam musíte najít Doubka. Je to můj starý známý a ten vám poradí, kudy dál."

Chvíli to trvalo, než se temnému mágovi podařilo skrýt z tváře výraz překvapení, ale nakonec se sebral a odpověděl: „Hmm, to by nám velice pomohlo. A jak toho Doubka mezi všemi těmi stromy najdeme?"

„To je snadná pomoc. Doubek je pátý nejvyšší strom na louce."

Temný mág poděkoval a vydal se směrem, který mu strom naznačil mávnutím větví. Po delší chvíli dorazili na louku. Byla velká a přetékala slunečním světlem. Chvíli jen tak stáli na okraji a mžourali. Když jejich oči přivykly, spatřili uprostřed louky dvě řady stromů. Obě byly seřazené od největšího stromu po nejmenší.

„Tak, jdeme najít Doubka!" zavelel temný mág a vykročil kupředu.

20-2-2 Setříděné stromy — 10 bodů

Na rozdíl od temného mága si my můžeme úlohu trochu zobecnit. Nebudeme hledat pátý, ale obecně k-tý nejvyšší strom.

Máme tedy dvě řady stromů, které jsou obě setříděné podle velikosti, a chceme nalézt k-tý nejvyšší z jejich sjednocení. Porovnání výšek dvou stromů je pochopitelně složitá operace, takže bychom ji chtěli použít co nejméněkrát.

Temný mág samozřejmě pospíchá, takže lineární řešení (slévání posloupností) je příliš pomalé.

Příklad: Mějme posloupnosti 12,8,3,1 a 20,19,15,4,2. Pátý největší prvek z jejich sjednocení je 8.

Mág si stoupl před strom, který se zdál být pátý nejvyšší z obou řad: „Strom Doubek, předpokládám."

„Ale ano! A vy račte být?" sklonil k němu strom svoji košatou korunu.

„Já jsem temný mág! Bydlím v temné věži na druhé straně řeky a právě teď jsem tu trochu zabloudil," vysvětloval mág.

„To chápu, vy lidé se neustále někde ztrácíte. Ale nic si z toho nedělejte, já vás zase najdu," prohlásil Doubek veselým tónem a vysadil si kořeny ze země. „Pojďte za mnou!"

Doubek byl na strom velice rychlý a poměrně vzdělaný. Mág měl co dělat, aby s ním udržel krok jak po cestě, tak v konverzaci. Asi po hodině svižné chůze dorazili na cestu, která procházela hvozdem. Doubek se zastavil a z koruny vyklepal několik veverek, havrana a spícího kocoura.

„Tudy se dostanete ven z hvozdu," ukázal jim cestu mávnutím větví. A než mu stačil mág poděkovat, Doubek se otočil a po několika krocích zmizel mezi ostatními stromy.

Teď už cesta ubíhala pohodlně. Druhý den začal hvozd řídnout a okolo poledne se před nimi otevřela rovná krajina. Na večer se utábořili a když druhý den opět vyrazili, spatřili v dálce před sebou malé městečko. Temný mág stejně neměl přesnou představu, kde by měl hledat temný kámen, a tak se rozhodl, že návštěva města nemůže být na škodu.

Když se k městu přiblížili, potkali na cestě vůz naložený senem. Táhli ho dva statní oři a na kozlíku seděl drobný človíček. Když uviděl povedenou družinku, otevřel ústa dokořán, až mu z nich vypadla fajfka.

„Tak co, strejdo? Uhneš nám, nebo tady na sebe budeme čumět do soudného dne?" nadhodil konverzačním tónem Felix a ladně vyskočil na kozlík.

„Krá-krá, kráááááá!" přisadil si Kiri a napůl přistál, napůl se zřítil do kupky sena.

Majitel povozu vzal nohy na ramena a přitom křičel, jako když ho na nože berou.

„Tak, to bychom měli," prohlásil spokojeně kocour a uvelebil se na seně. „Je libo svezení?"

Temný mág si povzdechl. Nějak mu nepřipadalo správné děsit obyčejné lidi. Na druhou stranu to alespoň vylepší jeho image. Ovšem ten vůz musí vrátit jeho majiteli. Je přece mág, ne nějaký špinavý zloděj.

Nasedli na vůz, Vilda si sedl na kozlík a pobídl koně. Už byli skoro ve městě, když si všimli, že se něco děje. Lidé pobíhali zmateně sem a tam a většina spěšně město opouštěla. Kdokoli se na ně podíval, odhodil všechny věci a s křikem na rtech utíkal, seč mu nohy stačily. Být temným mágem má i své stinné stránky. Například se nikoho nemůžete zeptat na cestu, protože než dokončíte otázku, dotyčný je dávno pryč a ještě přitom křičí nesrozumitelná slova.

Na náměstí se šikovaly stráže a jejich kapitán právě vysílal poštovního holuba...

20-2-3 Morseovkabezoddělovačů — 10 bodů

Interní vojenská sdělení se zásadně šifrují morseovkou. Kapitán ovšem ve zmatku zašifroval zprávu tak, že mezi písmeny a slovy zapomněl dělat oddělovače. Poštovní holub má namířeno do sousedního městečka, ve kterém je další vojenská posádka. Otázkou zůstává, jestli se jim vůbec podaří vzkaz rozluštit. Pomůžete jim?

Máte rozluštit text zakódovaný morseovkou bez oddělovačů. K dispozici máte slovník vojenských termínů (tj. slov, která mohou být v textu použita). Vašim cílem je vypsat překlad, který má nejméně slov. Pokud je možných nejkratších překladů víc, vypište libovolný z nich.

Příklad: Slovník obsahuje slova `attack`, `diet`, `indemnify`, `knights`, `pig`, `send`, `sets`, `the`, `zombie` a holub přinesl následující zprávu: `....-.-..-.....-.--...--.....-...`

Správný překlad je: „`send the knights`“. Zprávu je ještě možné přeložit jako „`send diet pig sets`“, avšak takový překlad je o slovo delší, takže to není správné řešení.

Pozn: Omlouváme se všem češtinářským puritánům za příklad s anglickými slovíčky. Bohužel se nám nepodařilo nalézt vhodný a rozumně krátký příklad v naší mateřštině.

Posádka se sešikovala a s rozklepanými koleny zírala, jak uprostřed náměstí zastavil vůz řízený zombií a přitom se ozvalo zlověstné zakrákání, které z celé situace udělalo jedno velké klišé. Z vozu neohrabaně slezla postava celá v černém. Kapitán polkl naprázdno. Teď začnou komplikace. Jednomu ze strážných vypadla z roztřesené ruky halapartna a zařinčela na dláždění.

„Jménem města –“ vypravil ze sebe chvějícím se hlasem kapitán, ale mág ho zastavil pozdvihnutou rukou.

„Ale kapitáne, nechcete dělat žádná ukvapená rozhodnutí, že ne?“ zašvitořil mág medovým hlasem. „Což takhle probrat vzniklou situaci nad šálkem dobrého černého čaje? A vaši muži si zatím mohou dát pauzu, vypadají trochu napjatě.“

Kapitán byl jako omámený. Pomalu se otočil ke své jednotce a zavelel pohov. Pak se i s temným mágem odporoučel do své kanceláře. Na náměstí zůstaly stráže ve velmi nejisté situaci a opatrně pozorovaly vůz s Vildou. Ze sena vyskočil Felix, protáhl si kočičí hřbet a ladným krokem se začal procházet. Přišel k jednomu strážnému, posadil se těsně před něho a naklonil hlavu ke straně. Strážný si ho prohlížel s napjatým výrazem.

„Baf!“ pronesl Felix suše.

Na dláždění zařinčelo několik halaparten a celá posádka se ztratila za dupotu sandálů v obláčku prachu.

„Ani se nerozloučili,“ postěžoval si Felix a začal se mýt.

Temný mág se po chvíli vrátil i s kapitánem, který měl na tváři nepřítomný výraz.

„Dobrá zpráva. Kapitán nám nejen dovolil zůstat ve městě, jak dlouho budeme chtít, ale také byl tak laskav a prozradil mi, kde se nachází místní knihovna.“

Knihovna nebyla velká a očividně v ní už dlouho nikdo neuklízel. Po zemi se válely nejrůznější knihy a svitky. Police byly prožrané červotoči a pořádně zaprášené. Zkrátka když jste si při pohledu na tuhle spoušť představili slovo úklid, první, co vás napadlo jako asociace, byl krumpáč.

Mág smutně pokýval hlavou. „Nejprve bychom měli – Felixi!! Co to tam děláš?!“

„Copak můžu za to, že tu není žádná bedýnka s pískem?“ opáčil kocour ublíženě. „A navíc je tu stejně nepořádek!“

20-2-4 Slovíčkaření — 10 bodů

Mág s Vildou potřebují uklidit v knihovně. To mimo jiné znamená, že musí setřídit všechny knížky podle názvů. Jak jste si již v našem příběhu zvykli, času nemají hrdinové nazbyt, takže by to potřebovali udělat opravdu rychle.

Máte dánu množinu slov – názvů knížek – a chcete ji setřídit podle abecedy. Všechna slova jsou zapsána pomocí konečné uspořádané abecedy Σ. Její velikost $|\Sigma|$ značme A. Dvě písmenka z abecedy můžete porovnat v konstantním čase, ale nezapomeňte, že porovnání dvou slov už není konstantní operace, její časová složitost je lineární k délce kratšího ze slov.

Snažte se o co nejmenší časovou složitost nejen vzhledem k počtu slov N, ale také vzhledem k jejich délce (součet délek všech slov si označme P) a k velikosti abecedy (A).

Příklad: Potřebujeme setřídit slova nad českou abecedou: kuchařka, zpěvník, zahradničení a necrotelecomicron. Vzestupně setříděná budou v pořadí: kuchařka, necrotelecomicron, zahradničení a zpěvník.

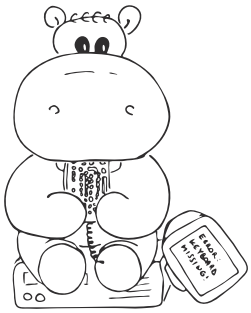
Uklízení v knihovně jim zabralo pěkných pár dní. Naštěstí mág znal několik užitečných kouzel, která pomohla věci urychlit, a stráže už je vůbec neobtěžovaly.

„Á, konečně jsem to našel,“ zvolal mág nadšeně a vítězoslavně pozvedl knihu nad hlavu. „Tady se píše, že se musíme vydat směrem . . . “

To be continued. . .

20-2-5 Hroší ohrádka — 10 bodů

Milí řešitelé a řešitelky.



Opět se zde setkáváme nad praktickou úložkou. Způsob odevzdávání a všechny ostatní detaily jsou stejné jako v minulé sérii. Takže pokud jste zapomněli, jak to v praktické úložce chodí, nebo jste se do řešení KSP zapojili teprve teď, podívejte se na úložku 20-1-5 z minulé série, kde naleznete potřebné informace.

Zadání: V jedné nejmenované zoo řeší problém. Přemnožili se jim hroši a je potřeba pro ně postavit nový výběh. Zoo hodlá využít opuštěný výběh po bobří rodince, která emigrovala do zahraničí při poslední povodni. Na pozemku se nachází spousta kůlů, které lze využít jako základ oplocení, avšak samotný plot už je zničený. Zoo se proto rozhodlo oplotit co největší možnou plochu tak, že použije stávající kůly a mezi nimi napne pletivo. Zůstává ovšem otázka, jak to provést – a to už je úkol pro vás.

Váš program dostane na vstupu seznam souřadnic kůlů v souboru `kuly.in`. Na prvním řádku je jediné číslo N, které představuje počet kůlů. Na následujících N řádcích se nacházejí souřadnice jednotlivých kůlů popsané jako dvojice čísel x_i, y_i. Souřadnice jsou celočíselné a vejdou se do 32-bitového integeru.

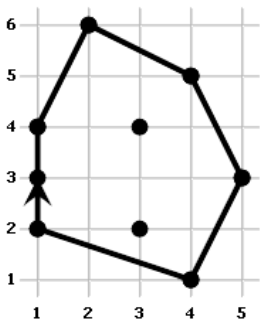
Nedá příliš přemýšlení, že největší plochu bude mít právě konvexní obal zadaných bodů. Váš program má za úkol tento obal spočítat a vypsat do souboru `plot.out` v následujícím formátu:

Na prvním řádku bude jediné číslo K. Na následujících K řádcích se nachází seznam hraničních kůlů, mezi kterými je plot natažen. Kůly musí být vypsány ve správném pořadí (tak, jak tvoří plot). Jako první musí být vypsán kůl s nejmenší x-ovou souřadnicí (pokud je takových kůlů víc, vyberte ten, který má zároveň nejmenší y-ovou souřadnici). Pokud si navíc kůly představíme zakreslené v rovině (tak, že x-ová osa roste doprava a y-ová roste vzhůru), pak musí být kůly vypsány po směru hodinových ručiček. Pro bližší pochopení se podívejte na následující příklad.

Příklad:

kuly.in

```
9
3 2
1 2
4 1
3 4
1 3
1 4
4 5
2 6
5 3
```



plot.out

```
7
1 2
1 3
1 4
2 6
4 5
5 3
4 1
```

Hint: Nepoužívejte reálná čísla či složité matematické funkce. Souřadnice jsou celočíselné, takže si celou dobu vystačíte s celými čísly. Pokud nevěříte, vezměte si k ruce matematicko-fyzikální tabulky (nebo podobnou literaturu).

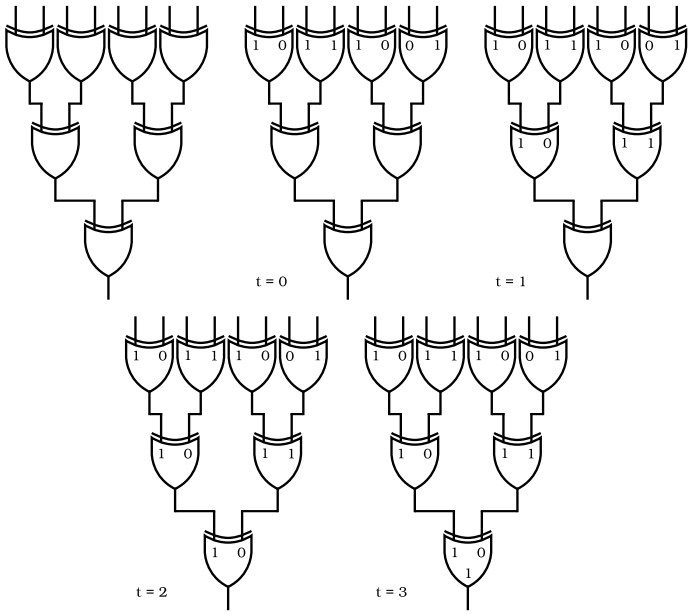
Pozn: Můžete předpokládat, že obsah konvexního obalu je vždy nenulový. Tj. nenastane případ, že by všechny kůly ležely na jedné přímce.

20-2-6 Hrady, hrádky, hradla — 11 bodů

Milí řešitelé, minule jsme si povídali o věcech takřka tajemných a nahlédli jsme do vnitřností vašich počítačů. Zjistili jsme, jak fungují ty nejmenší části mikroprocesoru, a pokusili jsme je alespoň trochu ovládnout. V tomto dílu našeho seriálu se spolu podíváme, jak rychle a efektivně naše obvody vlastně fungují.

Rychlostí zde míníme funkci závislosti času stráveného výpočtem, na velikosti vstupu. Pokud vám to připomíná časovou složitost algoritmu, jste na správné stopě. Stačí nám říct, že doba průchodu signálu hradlem, tj. doba, za kterou se po připojení signálů na vstup hradla nastaví na jeho výstupu správný výsledek operace, je jednotková. Poté si můžeme přestavit, že výpočet probíhá po hladinách. Pod pojmem „hladina“ si nemusíme představovat nic složitého, je to několik hradel, ke kterým výpočet dorazí ve stejné chvíli. Výpočet bude tedy probíhat tak, že v čase $t = 0$ se na vstup zapojení nastaví vstupní data. V čase $t = 1$ budou výsledky na výstupech hradel, která počítají výsledek ze vstupních dat. V čase $t = 2$ budou výsledky na výstupech hradel, která počítají výsledky z hradel předchozích dvou hladin a tak dále, až se spočítají výsledky na poslední hladině, která vydá konečný výsledek celého obvodu.

Abychom si ještě lépe rozuměli, připravili jsme si pro vás jednoduchý obvod, který zjišťuje, zda je na vstupu lichý či sudý počet jedniček. Na obrázku je nakreslen pro $n = 8$:



Zapojení funguje jednoduše: počítá si pro každou dvojici čísel ze vstupu, zdali je v ní počet jedniček lichý nebo ne. Následně provádí stejný výpočet, ovšem pro již vypočítané mezivýsledky, kterých je jenom polovina, a tak dále, až všechny mezivýsledky spojí do jediného výsledku. Takto přímočaře to sice funguje jen tehdy, je-li velikost vstupu mocnina dvojky (tedy $n = 2^k$ pro nějaké k), ale i kdyby nebyla, můžeme si vstup doplnit nulami na nejbližší vyšší mocninu dvojky, čímž výsledek nezměníme a n maximálně zdvojnásobíme.

Hladiny obvodu odpovídají „patrům“ na obrázku. Na první hladině leží $n/2$ hradel, na druhé $n/4$, ..., na i-té $n/2^i$, až na k-té jich je $n/2^k = n/n = 1$. Čas strávený výpočtem proto je $k = \log_2 n$. Naše zapojení tedy má logaritmickou časovou složitost.

Výrazem *efektivně* zde míníme hlavně spotřebu energetickou. Každá součástka v obvodu spotřebovává energii. Pro naše účely budeme předpokládat, že hradlo spotřebovává jednu jednotku energie, tj. všechny typy hradel jsou stejně náročné. Bude nás tedy zajímat funkce vyjadřující závislost počtu hradel na velikosti úlohy. Opět si můžeme všimnout, že je daná definice skoro povědomá a může připomínat definici paměťové složitosti algoritmu.

Spočítejme si hradla v našem příkladu po hladinách od nejnižší k nejvyšší. Celkem jich je $1+2+4+\ldots+n/4+n/2 = \sum_{k=0}^{\log_2 n-1} 2^k$. Součet řady můžete zjistit přímočaře ze vzorce pro součet geometrické řady. Napovíme, že se vám bude hodit vztah $x = a^{\log_a x}$. Pokud vám vyjde složitost lineární, a to $n - 1$, došli jste k správnému výsledku.

Tady naše povídání končí a abyste i vy dostali svůj prostor, následuje **zadání úložek:**

1. Navrhněte obvod, který bude počítat „hromadný OR“. Tedy trochu formálněji: na vstupu máme n bitů a na výstupu máme jednu hodnotu, která je 1 právě tehdy, je-li mezi vstupními bity aspoň jedna jednička. [4 body]

2. Dokažte, že vaše řešení první úlohy je nejlepší možné, čili že lepší časové ani paměťové složitosti už dosáhnout nejde. [7 bodů]

Recepty z programátorské kuchařky

I letos Vám kromě úloh budeme servírovat také recepty z programátorské kuchařky. Některé si vypůjčíme z dřívějších ročníků KSP, ale i k těm se budeme snažit připsat něco nového, aby si i zkušenější řešitelé přišli na své. V letošní první kuchařce si povíme o třídících algoritmech. Co to znamená? Pojem *třídění* je možná maličko nepřesný, nehodláme data (čísla, záznamy, řetězce a jiné) rozdělovat do nějakých tříd, ale přerovnat je do správného pořadí, protože se seřazenými údaji se mnohem lépe pracuje, například pokud v nich pak potřebujeme vyhledávat. Takové uspořádávání dat je denním chlebem každého programátora, a tak není divu, že třídící algoritmy jsou jedny z nejstudovanějších. My však nebudeme do nějakých velkých detailů a specialit příliš zabíhat. Zkrátka a dobře – budeme chtít třídit údaje rychle, úsporně a radostně.

Obvykle třídíme exempláře datové struktury typu pascalského záznamu. V takové datové struktuře bývá obsažena jedna význačná položka, *klíč*, podle které se záznamy řadí. Malinko si náš život zjednodušíme a budeme předpokládat, že třídíme záznamy obsahující pouze klíč, který je navíc celočíselný – budeme tedy třídit pole celých čísel. Vzhledem k počtu tříděných čísel N pak budeme vyjadřovat časovou (a paměťovou) složitost jednotlivých algoritmů, které si předvedeme.

Metody třídění můžeme rozdělit do dvou hlavních skupin, a to na *vnitřní třídění,* kdy si můžeme dovolit všechna data

načíst do (rychlé) paměti počítače, a na *vnější třídění,* kdy již třídění musíme realizovat opakovaným čtením a vytvářením diskových souborů. V tomto dílu se omezíme pouze na algoritmy vnitřního třídění a tříděné pole si nadeklarujeme takto:

```
const N = 100;
type Pole = array[1..N] of integer;
```

Nejjednodušší třídící algoritmy patří do skupiny *přímých metod.* Všechny mají několik společných rysů: Jsou krátké, jednoduché a třídí přímo v poli (nepotřebujeme pomocné pole). Tyto algoritmy mají většinou časovou složitost $\mathcal{O}(N^2)$. Z toho vyplývá, že jsou použitelné tehdy, když tříděných dat není příliš mnoho. Na druhou stranu pokud je dat opravdu málo, je zbytečně složité používat některý z komplikovanějších algoritmů, které si předvedeme později.

Stručně si přiblížíme tři nejznámější přímé algoritmy. *Třídění přímým výběrem (SelectSort)* je založeno na opakovaném vybírání nejmenšího čísla z dosud nesetříděných čísel. Nalezené číslo prohodíme s prvkem na začátku pole a postup opakujeme, tentokrát s nejmenším číslem na indexech $2, \ldots, N$, které prohodíme s druhým prvkem v poli. Poté postup opakujeme s prvky s indexy $3, \ldots, N$, atd. Je snadné si uvědomit, že když takto postupně vybíráme minimum z menších a menších intervalů, setřídíme celé pole (v i-tém kroku nalezneme i-tý nejmenší prvek a zařadíme ho v poli na pozici s indexem i).

```
procedure SelectSort(var A: Pole);
var i,j,k,x: integer;
begin
  for i:=1 to N-1 do
    begin
      k:=i;
      for j:=i+1 to N do
        if A[j]<A[k] then k:=j;
      x:=A[k]; A[k]:=A[i]; A[i]:=x;
    end;
end;
```

Pro úplnost si ještě řekněme pár slov o časové složitosti právě popsaného algoritmu. V i-tém kroku musíme nalézt minimum z $N-i+1$ čísel, na což spotřebujeme čas $\mathcal{O}(N-i+1)$. Ve všech krocích dohromady tedy spotřebujeme čas $\mathcal{O}(N+(N-1)+\ldots+3+2+1) = \mathcal{O}(N^2)$.

Třídění přímým vkládáním (InsertSort) funguje na podobném principu jako třídění přímým výběrem. Na začátku pole vytváříme správně utříděnou posloupnost, kterou postupně rozšiřujeme. Na začátku i-tého kroku má tato utříděná posloupnost délku $i-1$. V i-tém kroku určíme pozici i-tého čísla v dosud utříděné posloupnosti a zařadíme ho do utříděné posloupnosti (zbytek utříděné posloupnosti se posune o jednu pozici doprava). Není těžké si rozmyslet, že každý krok lze provést v čase $\mathcal{O}(N)$. Protože počet kroků algoritmu je N, celková časová složitost právě popsaného algoritmu je opět $\mathcal{O}(N^2)$.

```
procedure InsertSort(var A: Pole);
var i,j,x: integer;
begin
  for i:=2 to N do
    begin
      x:=A[i];
      j:=i-1;
      while (j>0) and (x<A[j]) do
        begin
          A[j+1]:=A[j];
          j:=j-1;
        end;
      A[j+1]:=x;
    end;
end;
```

(Upozornění: v našich příkladech předpokládáme, že máme v překladači zapnuto tzv. zkrácené vyhodnocování logických výrazu, třeba v předchozím while-cyklu se při `j=0` hodnoty `x` a `A[0]` již neporovnávají.)

Bublinkové třídění (BubbleSort) pracuje jinak než dva dříve popsané algoritmy. Algoritmu se říká „bublinkový“, protože podobně jako bublinky v limonádě „stoupají“ vysoká čísla v poli vzhůru. Postupně se porovnávají dvojice sousedních prvků, řekněme zleva doprava, a pokud v porovnávané dvojici následuje menší číslo po větším, tak se tato dvě čísla prohodí. Celý postup opakujeme, dokud probíhají nějaké výměny. Protože algoritmus skončí, když nedojde k žádné výměně, je pole na konci algoritmu setříděné.

```
procedure BubbleSort(var A: Pole);
var i,x: integer;
    zmena: boolean;
begin
  repeat
    zmena:=false;
    for i:=1 to N-1 do
      if A[i] > A[i+1] then
        begin
          x:=A[i]; A[i]:=A[i+1]; A[i+1]:=x;
          zmena:=true;
        end;
  until not zmena;
end;
```

Správnost algoritmu nahlédneme tak, že si uvědomíme, že po i průchodech while-cyklem bude posledních i prvků obsahovat největších i prvků setříděných od nejmenšího po největší (rozmyslete si, proč tomu tak je). Popsaný algoritmus se tedy zastaví po nejvýše N průchodech a jeho celková časová složitost v nejhorším případě je $\mathcal{O}(N^2)$, neboť na každý průchod spotřebuje čas $\mathcal{O}(N)$. Výhodou tohoto algoritmu oproti předchozím dvěma je, že je tím rychlejší, čím blíže bylo zadané pole k setříděnému stavu – pokud bylo úplně setříděné, tehdy algoritmus spotřebuje jen lineární čas, $\mathcal{O}(N)$.

Sofistikovanější třídící algoritmy pracují v čase $\mathcal{O}(N \log N)$. Jedním z nich je *Třídění slévánim (MergeSort),* založené na principu slévání (spojování) již setříděných posloupností dohromady. Představme si, že již máme dvě setříděné posloupnosti a chceme je spojit dohromady. Jednoduše stačí porovnávat nejmenší prvky obou posloupností a menší z těchto prvků vždy odstranit a přesunout do nové posloupnosti. Je zřejmé, že ke slití dvou posloupností potřebujeme čas úměrný součtu jejich délek.

My si zde popíšeme a předvedeme modifikaci algoritmu MergeSort, která používá pomocné pole. Algoritmus lze implementovat při zachování časové složitosti i bez pomocného pole, ale je to o dost pracnější. Existuje též modifikace algoritmu, která má počet fází (viz dále) v nejhorším případě $\mathcal{O}(\log N)$, ale pokud je již pole na začátku setříděné, proběhne pouze jediná a v takovém případě má algoritmus časovou složitost $\mathcal{O}(N)$. My si však zatajíme i tuto variantu.

Algoritmus pracuje v několika *fázích.* Na začátku první fáze tvoří každý prvek jednoprvkovou setříděnou posloupnost a obecně na začátku i-té fázi budou mít setříděné posloupnosti délky 2^{i-1}. V i-té fázi tedy vždy ze dvou sousedních 2^{i-1}-prvkových posloupností vytvoříme jedinou délky 2^i. Pokud N není násobkem 2^i, bude délka poslední posloupnosti zbytek po dělení N číslem 2^i. Zastavíme se, pokud $2^i \geq N$, tj. po $\lceil \log_2 N \rceil$ fázích. Protože v i-té fázi slijeme $\lceil N/2^i \rceil$ dvojic nejvýše 2^{i-1}-prvkových posloupností, je časová složitost jedné fáze $\mathcal{O}(N)$. Celková časová složitost popsaného algoritmu je pak $\mathcal{O}(N \log N)$.

```
procedure MergeSort(var A: Pole);
var P: Pole;        { pomocné pole }
    delka:integer;  { délka setříděných posl. }
    i: integer;     { index do vytvářené posl. }
    i1,i2: integer; { index do slévaných posl. }
    k1,k2: integer; { konce slévaných posl. }
begin
  delka:=1;
  while delka<N do
    begin
      i1:=1; i2:=delka+1; i:=1;
      k1:=delka; k2:=2*delka;
      while i<=N do
        begin
          { sléváme A[i1..k1] s A[i2..k2] }
          if k2>N then k2:=N;
          while (i1<=k1) or (i2<=k2) do
            if (i2>k2) or
               ((i1<=k1) and (A[i1]<=A[i2]))
            then
              begin
                P[i]:=A[i1]; i:=i+1; i1:=i1+1;
              end
            else
              begin
                P[i]:=A[i2]; i:=i+1; i2:=i2+1;
              end;
          i1:=k2+1; i2:=i1+delka;
          k1:=k2+delka;
          k2:=k2+2*delka;
        end;
      A:=P;
      delka:=2*delka;
    end;
end;
```

V čase $\mathcal{O}(N \log N)$ pracuje také algoritmus jménem *QuickSort.* Tento algoritmus je založen na metodě Rozděl a panuj. Nejprve si zvolíme nějaké číslo, kterému budeme říkat *pivot.* Více si o jeho volbě povíme později. Poté pole přeuspořádáme a rozdělíme je na dvě části tak, že žádný prvek v první části nebude větší než pivot a žádný prvek v druhé části naopak menší. Prvky v obou částech pak setřídíme rekurzivním zavoláním téhož algoritmu. Musíme ale dát pozor, aby byly v každém kroku obě části neprázdné (a rekurze tedy byla konečná). Je zřejmé, že po skončení algoritmu bude pole setříděné.

Malá zrada spočívá ve volbě pivota. Pro naše účely by se hodilo, aby po přeházení prvků levá i pravá část pole byly přibližně stejně velké. Nejlepší volbou pivota by tedy byl *medián* tříděného úseku, tj. prvek takový, jenž by byl v setříděném poli přesně uprostřed. Přeuspořádání jistě zvládneme v lineárním čase a pokud by pivoty na všech úrovních byly mediány, pak by počet úrovní rekurze byl $\mathcal{O}(\log N)$ a celková časová složitost $\mathcal{O}(N \log N)$ (na každé úrovni rekurze je součet délek tříděných posloupnosti nejvýše N). Ačkoli existuje algoritmus, který medián pole nalezne v čase $\mathcal{O}(N)$, v QuickSortu se obvykle nepoužívá, jelikož konstanta u členu N je příliš velká v porovnání s pravděpodobností, že náhodná volba pivota algoritmus příliš zpomalí. Většinou se pivot volí náhodně z dosud nesetříděného úseku – zkrátka se sáhne někam do pole a nalezený prvek se prohlásí za pivot. Dá se ukázat, že takovýto algoritmus s velmi vysokou pravděpodobností poběží v čase $\mathcal{O}(N \log N)$. Důkaz tohoto tvrzení je trošičku trikový a lze jej nalézt např. v knize Kapitoly z diskrétní matematiky od pánů Matouška a Nešetřila. Je však třeba si pamatovat, že pokud se pivot volí náhodně, může rekurze dosáhnout hloubky N a časová složitost algoritmu až $\mathcal{O}(N^2)$ – představme si, že se pivot v každém rekurzivním volání nešťastně zvolí jako největší prvek z tříděného úseku. V naší implementaci QuickSortu pro názornost nebudeme pivot volit náhodně, ale vždy jako pivot vybereme prostřední prvek tříděného úseku.

```
procedure QuickSort(var A:Pole; l,r:integer);
var i,j,k,x: integer;
begin
  i:=l; j:=r;
  k:=A[(i+j) div 2];
  repeat
    while A[i]<k do i:=i+1;
    while A[j]>k do j:=j-1;
    if i<=j then
      begin
        x:=A[i]; A[i]:=A[j]; A[j]:=x;
        i:=i+1;
        j:=j-1;
      end;
  until i >= j;
  if j>l then QuickSort(A, l, j);
  if i<r then QuickSort(A, i, r);
end;
```

Ještě si předvedeme dva třídící algoritmy, které jsou vhodné, pokud tříděné objekty mají některé další speciální vlastnosti. Prvním z nich je *třídění počítáním (CountSort).* To lze použít, pokud tříděné objekty obsahují pouze klíče a možných hodnot klíčů je málo. Tehdy si stačí spočítat, kolikrát se který klíč vyskytuje, a místo třídění vytvořit celé pole znovu na základě toho, kolik jednotlivých objektů obsahovalo pole původní. My si tento algoritmus předvedeme na příkladu třídění pole celých čísel z intervalu $\langle D, H \rangle$:

```
const D = 1;
      H = 10;
procedure CountSort(var A: Pole);
var C: array[D..H] of integer;
    i,j,k: integer;
begin
  for i:=D to H do C[i]:=0;
  for i:=1 to N do C[A[i]]:=C[A[i]] + 1;
  k:=1;
  for i:=D to H do
    for j:=1 to C[i] do
      begin
        A[k]:=i;
        k:=k+1;
      end;
end;
```

Časová složitost takovéhoto algoritmu je lineární v N, ale nesmíme zapomenout přičíst ještě velikost intervalu, ve kterém se prvky nacházejí ($K = H - D + 1$), protože nějaký čas spotřebujeme i na inicializaci pole počítadel. Celkem tedy $\mathcal{O}(N + K)$.

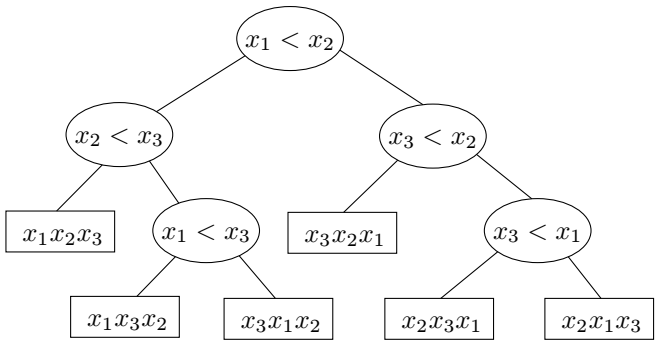
Pokud by tříděné objekty obsahovaly vedle klíčů i nějaká data, můžeme je místo pouhého počítání rozdělovat do přihrádek podle hodnoty klíče a pak je z přihrádek vysbírat v rostoucím pořadí klíčů. Tomuto algoritmu se říká *přihrádkové třídění (BucketSort)* a my si popíšeme jeho víceprůchodovou variantu *(RadixSort),* která je vhodnější pro větší hodnoty K. V první fázi si čísla rozdělíme do přihrádek (skupin) podle nejméně významné cifry a spojíme do jedné posloupnosti, v druhé fázi čísla roztřídíme podle druhé nejméně významné cifry a opět spojíme do jedné posloupnosti, atd. Je důležité, aby se uvnitř každé přihrádky zachovalo pořadí čísel v posloupnosti na začátku fáze, tj. posloupnost uložená v každé přihrádce je vybranou podposloupností posloupnosti ze začátku fáze. Tvrdíme, že na konci i-té fáze obsahuje výsledná posloupnost čísla utříděná podle i nejméně významných cifer. Zřejmě i-té nejméně významné cifry tvoří neklesající posloupnost, neboť podle nich jsme právě v této fázi rozdělovali čísla do přihrádek, a pokud dvě čísla mají tuto cifru stejnou, jsou uložena v pořadí dle jejich $i - 1$ nejméně významných cifer, neboť v každé přihrádce jsme zachovali pořadí čísel z konce minulé fáze. Na závěr poznamenejme, že místo čísel podle cifer lze do přihrádek rozdělovat též textové řetězce podle jejich znaků, atp.

Jak je to s časovou složitostí této varianty RadixSortu? Pokud třídíme celá čísla od 1 do K a v každém kroku je rozdělujeme do ℓ přihrádek, potřebujeme $\log_\ell K$ průchodů (tolik je cifer v zápisu čísla K v ℓ-kové soustavě). Každý průchod spotřebuje čas $\mathcal{O}(N + \ell)$, takže celý algoritmus běží v čase $\mathcal{O}((N + \ell) \log_\ell K)$. To je $\mathcal{O}(N)$, pokud K a ℓ jsou konstanty. My si předvedeme implementaci algoritmu pro $K = 255$ a $\ell = 2$ (čísla budeme rozhazovat do přihrádek podle bitů v jejich binárním zápisu).

```
const K=255;
procedure RadixSort(var A: Pole);
var P0,P1: Pole;
    k1,k2: integer;
    i: integer;
    bit: integer;
begin
  bit:=1;
  while bit<=K do
    begin
      k1:=0; k2:=0;
      for i:=1 to N do
        if (A[i] and bit)=0 then
          begin
            k1:=k1+1; P0[k1]:=A[i];
          end
        else
          begin
            k2:=k2+1; P1[k2]:=A[i];
          end;
      for i:=1 to k1 do A[i]:=P0[i];
      for i:=1 to k2 do A[k1+i]:=P1[i];
      bit:=bit shl 1;
    end;
end;
```

Na závěr našeho povídání o třídících algoritmech si ukážeme, že třídit obecné údaje, se kterými neumíme provádět nic jiného, než je navzájem porovnávat, rychleji než $\mathcal{O}(N \log N)$ nejen nikdo neumí, ale také ani umět nemůže. Libovolný třídící algoritmus založený na porovnávání a prohazování prvků totiž musí na některé vstupy vynaložit řádově alespoň $N \log N$ kroků. (RadixSort na první pohled tento výsledek porušuje, na druhý však už ne, když si uvědomíme, o jak speciální druh tříděných dat se jedná.)

Třídící algoritmus v průběhu své činnosti nějak porovnává prvky a nějak je přehazuje. Provedeme myšlenkový experiment. Pozměníme algoritmus tak, že nejdříve bude pouze porovnávat, podle toho zjistí, jak jsou prvky v poli uspořádány, a když už si je jistý správným pořadím, prvky najednou popřehází. Tím se algoritmus zpomalí nejvýše konstanta-krát. Také pro jednoduchost předpokládejme, že všechny tříděné údaje jsou navzájem různé. Porovnávací činnost algoritmu si pak můžeme popsat tzv. *rozhodovacím stromem.* Zde je příklad rozhodovacího stromu pro tříprvkové pole:



Každý vrchol obsahuje porovnání dvou prvků x a y, v levém podstromu daného vrcholu je činnost algoritmu pokud $x < y$, v pravém podstromu činnost při $x \geq y$. V listech je už jisté správné pořadí prvků.

Každému algoritmu odpovídá nějaký rozhodovací strom a každý průběh činnosti algoritmu odpovídá průchodu rozhodovacím stromem od kořene do nějakého listu. Naším cílem bude ukázat, že v libovolném rozhodovacím stromu (a tedy i libovolném odpovídajícím algoritmu) bude existovat cesta z kořene do nějakého listu (neboli výpočet algoritmu) délky $N \log N$.

Kolik maximálně hladin h, a tedy i jaká nejdelší cesta se v takovém stromu může vyskytnout? Náš strom má tolik listů, kolik je možných pořadí tříděných prvků, tedy právě $N!$. Různým pořadím totiž musí odpovídat různé listy, jinak by algoritmus netřídil (předpokládáme přeci, že to, jak má prvky prohazovat, může zjistit jenom jejich porovnáváním), a naopak každé pořadí prvků jednoznačně určuje cestu do příslušného listu. Na nulté hladině je jediný vrchol, na každé další hladině se oproti předchozí počet vrcholů nejvýše zdvojnásobí, takže na i-té hladině se nachází nejvýše 2^i vrcholů. Proto je listů stromu nejvýše 2^h (některé listy mohou být i výše, ale za každý takový určitě chybí jeden vrchol na h-té hladině). Z toho víme, že platí:

$$2^h \geq \textit{počet listů} \geq N!,$$

a proto:

$$h \geq \log_2(N!).$$

Logaritmus faktorálu se těžko počítá přesně, ale můžeme si ho zdola odhadnout pomocí následujícího pozorování:

$$n! = \underbrace{n \cdot (n-1) \cdot \ldots \cdot (n/2)}_{n/2 \text{ členů, každý} \geq n/2} \cdot \ldots \geq$$

$$\geq (n/2)^{(n/2)}.$$

Dosazením získáme:

$$h \geq \log_2(N!) \geq \log_2((N/2)^{N/2}) =$$

$$= \frac{N}{2}\log_2(N/2) = \frac{1}{2}\cdot N(\log_2 N - 1) \geq \frac{1}{4}\cdot N\log_2 N.$$

Vidíme tedy, že pro každý třídící algoritmus existuje vstup, na kterém se bude muset provést alespoň $c \cdot N \log N$ kroků, kde $c > 0$ je nějaká konstanta.

Poznámky na okraj:

- Zkuste si též rozmyslet (drobnou modifikací předchozího důkazu), že ani *průměrná* časová složitost třídění nemůže být lepší než $N \log N$.
- Odvodit průměrnou složitost QuickSortu vlastně není zase tak těžké. Zkusme následující úvahu: Pokud by pivot nebyl přesně medián, ale alespoň se nacházel v prostřední třetině setříděného úseku, byla by složitost stále $\mathcal{O}(N \log N)$, jen by se zvýšila konstanta v $\mathcal{O}$-čku. Kdybychom pivot volili náhodně, ale po rozdělení prvků si zkontrolovali, jestli pivot padl do prostřední třetiny, a pokud ne (jeden z úseků by byl moc velký a druhý moc malý), volbu bychom opakovali, v průměru by nás to stálo konstantní počet pokusů (pozorování z řešení úlohy 16-1-5: pokud čekáme na událost, která nastává náhodně s pravděpodobností p, stojí nás to v průměru $1/p$ pokusů; zde je $p = 1/3$), takže celková složitost by v průměru vzrostla jen konstantně. Původní QuickSort sice žádné takové opakování volby neprovádí a rovnou se zavolá rekurzivně na velký i malý úsek, ale opět se po v průměru konstantně mnoha iteracích velký úsek zredukuje na nejvýše 2/3 původní velikosti a třídění malých úseků jednotlivě nezabere víc času, než kdyby se třídily dohromady.
- Kdybychom u QuickSortu použili rekurzivní volání jen na menší interval, zatímco ten větší bychom obsloužili přenastavením proměnných a skokem na začátek právě prováděné procedury, zredukovali bychom paměťovou složitost na $\mathcal{O}(\log N)$, jelikož každé další rekurzivní volání zpracovává alespoň dvakrát menší úsek než to předchozí. Časové složitosti tím však nepomůžeme.
- Počet přihrádek u RadixSortu vůbec nemusí být konstanta – pokud např. chcete třídit N čísel v rozsahu $1 \ldots N^k$, stačí si zvolit $\ell = N$ a fází bude jenom k. Pro pevné k tak dosáhneme lineární časové složitosti.

Dnešní menu Vám servírovali
Tomáš Valla, Martin Mareš a Dan Kráľ