Nalezení *K*-tého nejmenšího prvku

z daných *N* čísel ($1 \leq K \leq N$)

speciální případ: $K = N/2$ … ***medián***

1. Setřídit čísla v poli S

výsledek: S[*K*-1] O(*N*.log *N*)

2. Postavit z čísel v poli S haldu

O(*N*)

z ní *K*-krát odebrat minimum O(*K*.log *N*)

celkem časová složitost O(*N* + *K*.log *N*)

3. Postavit v poli haldu z prvních *N*-*K*+2 prvků

K-2 prvků nechat stranou

Pak opakujeme (*K*-2)-krát:

- odebrat minimum z haldy (není *K*-tý nejmenší, je menší!)
- zařadit místo něj jeden prvek do haldy

Nakonec z haldy odebereme dvakrát minimum
= prvky v pořadí (*K*-1)-tý a *K*-tý nejmenší.

Časová a paměťová úspora – pracujeme s menší haldou
(např. pro medián je halda přibližně poloviční
– což ale znamená, že má jen o 1 hladinu méně)

4. QuickSelect (modifikace QuickSortu)

Základní idea:

- po rozdělení seznamu podle pivota pracujeme dál jen s tou částí, ve které je hledaný prvek (poznáme ji podle délky)

```python
def quickselect(s, k):
    """výběr k-tého nejmenšího prvku ze seznamu s"""
    if len(s) <= 1: return s[0]
    x = s[len(s) // 2]
    vlevo = [ a for a in s if a < x ]
    stred = [ a for a in s if a == x ]
    vpravo = [ a for a in s if a > x ]
    a = len(vlevo)
    b = len(vlevo) + len(stred)

    if k < a:
        return quickselect(vlevo, k)
    elif k < b:
        return x
    else:
        return quickselect(vpravo, k-b)
```

Jiná implementace – nevytváří se nové seznamy:

- místo nových seznamů pracujeme s úseky původního seznamu
- po přerovnání prvků podle pivota pracujeme dál jen s tou částí, ve které je hledaný prvek, druhou část už není třeba dotřídit
- volba té správné části: ta, v níž leží *K*-tý nejmenší prvek, tzn. kde je index *K*-1
- výsledek výpočtu: prvek S[*K*-1]
- při realizaci není třeba rekurze ani zásobník, jediné rekurzivní volání se snadno nahradí jednoduchým cyklem

```
def quickselect(s, k):
    """výběr k-tého nejmenšího prvku ze seznamu s"""
    zac, kon = 0, len(s)-1
    while zac < kon:
        x = s[k-1]      #možná volba pivota, lze i jinak
        i, j = zac, kon
        while i <= j:
            while s[i] < x: i += 1
            while s[j] > x: j -= 1
            if i < j:
                s[i], s[j] = s[j], s[i]
                i += 1; j -= 1
            elif i == j:
                i += 1; j -= 1
        # úsek <zac,kon> rozdělen na <zac,j> a <i,kon>
        if k-1 < i: kon = j
        if k-1 > j: zac = i
    return s[k-1]
```

Časová složitost

- v nejhorším případě:
provede se plný QuickSort (jeho nejhorší případ), složitost **O(N^2)**

- v nejlepším případě (půlení):
procházejí se po řadě úseky délky N, N/2, N/4, …, 1,
celkem vykonaná práce = součet jejich délek se blíží k 2N,
tedy časová složitost algoritmu **O(N)**

- v průměrném případě rovněž **O(N)**

5. Lineární algoritmus

- obdobný postup jako v případě QuickSelectu,
jenom pivota, podle něhož rozdělujeme prvky v poli, nevolíme náhodně, ale nějak chytře – aby byl natolik blízký mediánu, že nám zaručí lineární časovou složitost **O(*N*)** i v nejhorším případě (ale nemusí to být pokaždé přesně medián)

- jedna možná taková volba pivota *X*: medián z mediánů pětic (zajišťuje i v nejhorším případě zahodit alespoň 3/10 prvků z aktuálního úseku a to stačí)

Další metody návrhu algoritmů

předvýpočet

hladové algoritmy

Předvýpočet

Místo přímého výpočtu
vstup → výsledek s časovou složitostí X

provedeme nejprve předvýpočet
vstup → předpočítaná data s časovou složitostí Y

a poté výsledný výpočet
předpočítaná data → výsledek s časovou složitostí Z

Přitom výsledná časová složitost Y + Z je (řádově) menší než X, takže tím celý výpočet urychlíme (zvýšíme časovou efektivitu). Zaplatíme za to vyššími paměťovými nároky – potřebujeme paměť navíc na uložení předpočítaných dat.

→ *výměna času za paměť*

Typické způsoby předvýpočtu

- setřídit data
(nebo jinak vhodně přerovnat, aby se nám s nimi dobře pracovalo, uložené hodnoty přitom neměníme)

- překódovat data
(nahradit komplikovanější údaje jednoduššími, aby se nám s nimi lépe pracovalo)

- přepočítat data
(nad vstupními daty provést vhodný výpočet a uložit si upravené údaje, které později při řešení úlohy budeme opakovaně využívat)

Příklad: nejčastější číslo

Je dána posloupnost čísel délky N, hodnoty se v ní mohou opakovat.
Které číslo má v posloupnosti nejvíce výskytů?

Přímé řešení:
primitivně spočítat výskyty každého čísla → časová složitost $O(N^2)$

Předvýpočet:
- čísla setřídíme v čase $O(N \log N)$
- výsledek pak určíme snadno při jednom průchodu

Příklad: časové údaje

V plánovacím kalendáři na tento rok máte záznamy o několika akcích, kterých se máte zúčastnit. Pro každou akci je uveden její začátek a konec ve tvaru čtyř celých čísel
„měsíc den hodina minuta“
... a zadání úlohy nějak pokračuje, řešíme třeba přejezdy mezi akcemi, časové kolize (výběr akcí, kterých se stihneme zúčastnit), apod.

Předvýpočet:
- pro úsporné uložení vstupních informací i pro rychlejší následné výpočty je vhodné překódovat začátek a konec každé akce do jediného celého čísla = kolikátá minuta v aktuálním roce to je

Příklad: slova v textu

Je dán text délky *N* slov (posloupnost *N* slov oddělených mezerami) a seznam *K* klíčových slov. Všechna slova v textu i klíčová slova mají délku nejvýše *L*. Určete délku nejkratšího (co do počtu slov) souvislého úseku textu obsahujícího všechna klíčová slova.

- opakované porovnávání slov je pomalé a pomocí slov také nelze indexovat pole, vyplatí se proto na začátku úlohu překódovat: klíčová slova označíme čísly od 1 do *K*, všechna ostatní slova označíme 0 (nemusíme je vůbec rozlišovat)

- úlohu tak převedeme do podoby: v posloupnosti *N* celých čísel (hodnoty od 0 do *K*) nalézt co nejkratší souvislý úsek obsahující každé z čísel 1..*K*

- pro překódování si postavíme trii z klíčových slov v čase O(*K.L*) a text pak převedeme při jednom průchodu v čase O(*N.L*)

Příklad: prefixové součty

Je dána posloupnost nul a jedniček délky *N*.
Dostáváme *K* dotazů na intervaly v této posloupnosti: Kolik jedniček tento interval obsahuje?

Přímé řešení:
pro každý dotaz projedeme interval délky O(*N*) a spočítáme v něm jedničky → celková časová složitost pro *K* dotazů je O(*K.N*)

Předvýpočet:

- k zadané posloupnosti čísel *a* spočítáme její prefixové součty

$$p_i = a_0 + a_1 + \ldots + a_i$$

- počítáme je podle vztahu $p_0 = a_0$

  $p_i = p_{i-1} + a_i$ pro $i > 0$

  … každý spočítáme v konstantním čase,
  takže časová složitost výpočtu všech prefixových součtů je O(N)

- p_i je součet prvních i členů posloupnosti *a*
  = počet jedniček v intervalu od začátku po *i*-tý prvek (včetně)

- každý dotaz na interval *x*, *y* zodpovíme v konstantním čase,
  výsledkem je $p_y - p_{x-1}$
  → časová složitost pro K dotazů je O(K)

Časová složitost celého řešení: O(N+K)

Příklad: 2D prefixové součty

Je dána matice nul a jedniček o rozměrech *R x S*.
Dostáváme *K* dotazů na obdélníkové výřezy v této matici: Kolik jedniček tento obdélník obsahuje?

Přímé řešení:
pro každý dotaz projedeme zkoumaný obdélník a spočítáme v něm jedničky → velikost obdélníka je O(*R.S*), proto celková časová složitost pro *K* dotazů je O(*K.R.S*)

Předvýpočet:

- k zadané matici čísel a spočítáme její 2D prefixové součty $p_{i,\,j}$ = součet prvků (neboli počet jedniček) v obdélníku s levým horním rohem [0, 0] a pravým dolním rohem [i, j]

- hodnoty $p_{i,j}$ počítáme po řádcích, v každém řádku zleva doprava:
  pro $i = 0$ nebo $j = 0$ … prefixový součet v řádku 0 resp. sloupci 0
  pro $i > 0$, $j > 0$ … $p_{i,\,j} = a_{i,\,j} + p_{i,\,j-1} + p_{i-1,\,j} - p_{i-1,\,j-1}$
  … každou spočítáme v konstantním čase,
  takže časová složitost výpočtu všech je O($R.S$)

- každý dotaz na obdélník [x, y], [u, v] zodpovíme v konstantním čase, výsledkem je $p_{u,\,v} - p_{x-1,\,v} - p_{u,\,y-1} + p_{x-1,\,y-1}$
  → časová složitost pro K dotazů je O(K)

Časová složitost celého řešení: O($R.S + K$)

Příklad: maximální jedničková podmatice

Je dána matice nul a jedniček o rozměrech $R \times S$. Určete velikost (tzn. plochu) maximálního obdélníkového výřezu tvořeného samými jedničkami.

Přímé řešení: zkoušení všech možností

- zkusíme postupně všechny obdélníkové výřezy – je jich $O(R^2.S^2)$
- každý zkontrolujeme, zda obsahuje samé jedničky – práce $O(R.S)$

→ proto celková časová složitost $O(R^3.S^3)$

Předvýpočet sloupců jedniček (prefixové součty):

- pro každou jedničku v matici spočítáme, kolik jedniček se nachází v souvislém sloupci pod ní:
  $p_{i,\,j} = k$, pokud prvek $[i, j]$ samotný a dalších k-1 prvků souvisle pod ním jsou jedničky, jinak $p_{i,\,j} = 0$

- hodnoty $p_{i,j}$ spočítáme po sloupcích vždy zdola nahoru (jako prefixové součty), každou spočítáme v konstantním čase

→ časová složitost celého předvýpočtu je O($R.S$)

- zkoušíme všechny možné levé horní rohy obdélníka (všechny jedničky), pro každý z nich postupně všechny pravé horní rohy (souvislá řada jedniček napravo od levého horního rohu), pro každý z nich již určíme plochu maximálního jedničkového obdélníka v konstantním čase

→ časová složitost výpočtu je O($R.S^2$)

Časová složitost celého řešení je také O($R.S^2$)

Předvýpočet obdélníků jedniček (2D prefixové součty):

- pro každý prvek [i, j] v matici spočítáme, kolik jedniček se nachází v obdélníku mezi [0, 0] a [i, j]
    - stejně jako v předchozí úloze, je to vlastně součet prvků, každou hodnotu spočítáme v konstantním čase

→ časová složitost celého předvýpočtu je O($R.S$)

- zkusíme postupně všechny obdélníkové výřezy – je jich O($R^2.S^2$)
- každý zkontrolujeme, zda obsahuje samé jedničky = jeho velikost je rovna počtu jeho jedniček

→ pro každý obdélník díky předvýpočtu konstantní práce

Časová složitost celého řešení je tedy O($R^2.S^2$)

Lepší předvýpočet – přiléhavé nuly:

- pro každou jedničku v matici spočítáme, kolik jedniček se nachází v souvislém sloupci pod ní a kolik nad ní
  … stejně jako v předchozím případě časová složitost O(*R.S*)

- pozorování: každá strana maximálního jedničkového obdélníka musí přiléhat k nějaké nule (nebo okraji matice)

- zkoušíme všechny možné jedničky z levého sloupce obdélníka, které přiléhají k nule (tzn. jedničky následující na řádku po nule)
- pro každou z nich postupně všechny pravé okraje obdélníka (souvislá řada jedniček směrem napravo)
- pro každý z nich již určíme plochu maximálního jedničkového obdélníka v konstantním čase (podobně jako minule, jenom počítáme maximum nahoru i dolů)

→ časová složitost výpočtu je O(*R.S*)

Časová složitost celého řešení je O(*R.S*)

Hladové algoritmy (Greedy Search)

- programovací technika pro řešení optimalizačních úloh

- v každém kroku vybírá lokální maximum (příp. minimum)
  s cílem dojít tak ke globálnímu maximu (minimu)

- v každém kroku chceme získat co nejvíce co se jenom dá
  bez ohledu na to, co nás ještě čeká

- funguje pouze u některých optimalizačních úloh, často nefunguje
  (najde nějaké dost dobré řešení, ale ne vždy to optimální)
  … musíme vždy ověřit a dokázat správnost

- pokud funguje, vede k jednoduchému a efektivnímu řešení

Příklad 1: platidla

Úkol: Jsou dány hodnoty používaných mincí, od každé máme k dispozici neomezený počet kusů. Zaplaťte danou částku *N* minimálním počtem mincí.

Řešení: V každém kroku použijeme maximální minci, jakou můžeme. Na zbývající částku aplikujeme stejný postup.

a) běžná sada mincí 1, 2, 5, 10, …
→ hladový algoritmus FUNGUJE,
ale musí se to pořádně dokázat!

b) sada mincí 1, 4, 5, 10
máme zaplatit částku 8 → hladový algoritmus určí CHYBNÉ ŘEŠENÍ
5+1+1+1 (tzn. 4 mince), optimální je použít 4+4 (2 mince)

c) sada mincí 3, 8
máme zaplatit částku 9 → hladový algoritmus NENAJDE ŘEŠENÍ,
ačkoliv existuje řešení 3+3+3

Příklad 2: minimální kostra grafu

Úkol: V hranově ohodnoceném grafu nalézt (jednu libovolnou) minimální kostru.

Řešení: Kruskalův algoritmus – hladový přístup:
zkoušíme přidat hrany do kostry v pořadí podle jejich velikosti

Příklad 3: přidělení pracovních úkolů

Úkol:
Zadané pracovní úkoly jsou popsány jako uzavřené intervaly na časové ose (je dán čas začátku a čas konce každého úkolu). Určete minimální počet pracovníků, kteří mohou všechny úkoly vykonat.

Na každém úkolu pracuje pouze jeden pracovník.
Každý pracovník může vykonat kterýkoliv z úkolů.
Pracovník může postupně vykonat více úkolů, ale nikdy dva zároveň.

Řešení (hladově):

Předpokládejme, že pracovníci jsou očíslováni 1, 2, 3, …

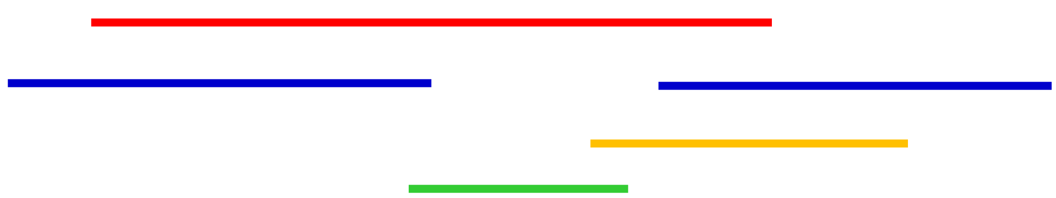
- všechny úkoly seřadíme podle času jejich začátku
- procházíme seřazené úkoly, každému úkolu vždy přiřadíme volného pracovníka s nejmenším číslem

Zdůvodnění: nového pracovníka *k* přidáme jenom tehdy, když všichni pracovníci 1, 2, …, *k*-1 právě plní úkol, tedy když ho nutně potřebujeme (protože máme *k* překrývajících se úkolů)

Chybné hladové řešení:

- všechny úkoly seřadíme podle jejich délky (sestupně)
- procházíme seřazené úkoly, každému úkolu vždy přiřadíme volného pracovníka s nejmenším číslem

Protipříklad: přidělení pracovníci 1 2 3 4
– algoritmus požaduje 4 pracovníky, přitom stačí 3



Příklad 4: problém batohu

Úkol:
Máme N předmětů, pro každý z nich známe hmotnost m_i a cenu p_i.
Známe kapacitu batohu M (jakou maximální hmotnost unese).

Úkolem je nalézt podmnožinu předmětů tak, aby součet jejich hmotností nepřesáhl M a součet jejich cen byl co největší.

Neúspěšné pokusy o hladové řešení:

a) seřadit předměty podle hmotnost vzestupně
= přednostně dávám do batohu lehké předměty, aby se jich tam vešlo co nejvíce

Protipříklad: N = 3, (10, 5), (15, 7), (30, 25), M = 30
Vyberu první dva předměty s celkovou cenou 12, ale lepší je vzít třetí předmět s cenou 25.

b) seřadit předměty podle ceny sestupně
= přednostně dávám do batohu drahé předměty, aby byla cena maximální

Protipříklad: N = 3, (30, 25), (15, 20), (10, 15), M = 30
Vyberu pouze první předmět s cenou 25, ale lepší je vzít druhý a třetí předmět s celkovou cenou 35.

c) seřadit předměty podle poměru cena/hmotnost sestupně
= přednostně dávám do batohu předměty s maximální „hustotou“ ceny (maximální cenou na jednotku hmotnosti)

Protipříklad: N = 3, (16, 16), (15, 14), (15, 13), M = 30
poměry cena/hmotnost: 16/16=1, 14/15, 13/15

Vyberu pouze první předmět s cenou 16, další se už do batohu nevejdou. Lepším řešením je ale vzít druhý a třetí předmět s horší „hustotou“ ceny, jejich celková cena je 27.

Správné řešení není hladové:

a) hrubou silou – vyzkoušet všechny možné výběry předmětů a spočítat jejich ceny

Takových výběrů je ovšem 2^{N}, takže se jedná o velmi neefektivní řešení s exponenciální časovou složitostí.

b) dynamické programování
- pokud jsou všechny hmotnosti celočíselné a známe přípustné rozmezí hodnot
- pseudopolynomiální řešení s časovou složitostí O($N.M$)