Sortowanie

Dane wejściowe: Ciąg n liczb $\langle a_1, a_2, \ldots, a_n \rangle$.

Wynik: Permutacja $\langle a'_1, a'_2, \ldots, a'_n \rangle$ ciągu wejściowego taka, że $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

Sortowanie

Dane wejściowe: Ciąg n liczb $\langle a_1, a_2, \ldots, a_n \rangle$.

Wynik: Permutacja $\langle a'_1, a'_2, \ldots, a'_n \rangle$ ciągu wejściowego taka, że $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

Wcześniej omówione algorytmy:

- `Insertion-Sort` – czas $\Theta(n^2)$.
- `Merge-Sort` – czas $\Theta(n \lg n)$.

Heapsort

Kopiec – drzewo binarne zrównoważone (może brakować prawej końcówki ostatniego poziomu).

Własność kopca: Wartość w każdym węźle jest niemniejsza niż wartości we wszystkich jego synach.

Heapsort

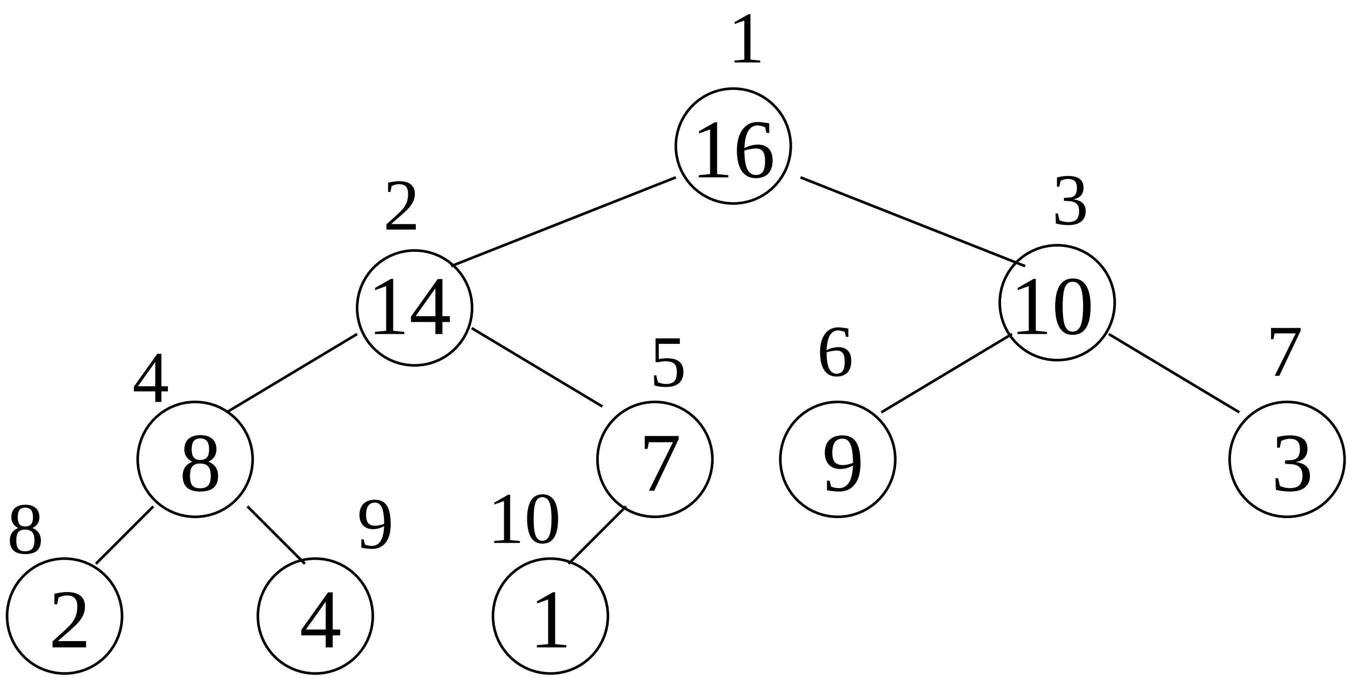
Kopiec – drzewo binarne zrównoważone (może brakować prawej końcówki ostatniego poziomu).

Własność kopca: Wartość w każdym węźle jest niemniejsza niż wartości we wszystkich jego synach.

Umieszczenie kopca w tablicy A (*length*$[A]$ - długość A, *heap-size*$[A]$ liczba elementów kopca w A):

- $A[1]$ – korzeń
- `Parent(`i`)`
  `return` $\lfloor i/2 \rfloor$
- `Left(`i`)`
  `return` $2i$
- `Right(`i`)`
  `return` $2i+1$

Kopiec



1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

wysokość węzła odległość od najdalszego liścia (liczba krawędzi).

Heapify

i - początkowa pozycja “wtapianego” elementu.

Poddrzewa `Left(`i`)` i `Right(`i`)` mają własność kopca.

`Heapify(`A`,`i`)`
$l \leftarrow$ `Left(`i`)`
$r \leftarrow$ `Right(`i`)`
`if` $l \leq$ *heap-size*`[`A`]` `and` $A[l] > A[i]$
    `then` *largest* $\leftarrow l$
    `else` *largest* $\leftarrow i$
`if` $r \leq$ *heap-size*`[`A`]` `and` $A[r] > A[$*largest*$]$
    `then` *largest* $\leftarrow r$
`if` *largest* $\neq i$ `then`
    `zamień` $A[i] \leftrightarrow A[$*largest*$]$
    `Heapify(`A`,`*largest*`)`

Po zakończeniu: Poddrzewo i ma własność kopca.

złożoność Heapify

- Niech n rozmiar drzewa o korzeniu i.
- Rozmiar większego z podrzew zaczepionych w synach i jest $\leq 2n/3$.
  (najgorszy przypadek: ostatni poziom wypełniony do połowy)
- Czas: $T(n) \leq \Theta(1) + T(2n/3)$.
  (wiersze 1–9 plus rekurencja)
- Z Tw. o rekurencji uniwersalnej (przypadek 2):

$$T(n) = O(\lg n)$$

Build-Heap

Początkowo A zawiera nieuporządkowany ciąg.

```
Build-Heap(A)
1 heap-size[A] ← length[A]
2 for i ← ⌊length[A]/2⌋ downto 1 do
3    Heapify(A,i)
  ▷ dla i > length[A]/2 - liście
```

Po zakończeniu A ma strukturę kopca.

Build-Heap

Początkowo A zawiera nieuporządkowany ciąg.

```
Build-Heap(A)
1 heap-size[A]←length[A]
2 for i ← ⌊length[A]/2⌋ downto 1 do
3   Heapify(A,i)
  ▷ dla i >length[A]/2 - liście
```

Po zakończeniu A ma strukturę kopca.

Liczba węzłów o wysokości h jest $\leq \lceil n/2^{h+1} \rceil$ (ćw.).

Czas:

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \lceil \frac{n}{2^{h+1}} \rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \right) = O(n)$$

bo: $\sum_{h=0}^{\infty} h/2^h = \frac{1/2}{(1-1/2)^2} = 2$ (*)

Uzasadnienie (*)

dla $|x| < 1$: $\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$

różniczkujemy: $\sum_{k=0}^{\infty} kx^{k-1} = \frac{1}{(1-x)^2}$

mnożymy przez x: $\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}$

dla $x = 1/2$: $\sum_{h=0}^{\infty} h(1/2)^h = \frac{1/2}{(1-1/2)^2}$

Heapsort

```
Heapsort(A)
  Build-Heap(A)
  for i ← length[A] downto 2 do
      zamień A[1] ↔ A[i]
      heap-size[A]←heap-size[A]−1
      Heapify(A,1)
```

Czas: $O(n\lg n)$, bo: `Build-Heap` – $O(n)$, oraz każde `Heapify` – $O(\lg n)$.

Kolejki priorytetowe

Operacje:

- `Insert(`S`,`x`)` $S \leftarrow S \cup \{x\}$
- `Maximum(`S`)` zwraca maximium z S
- `Extract-Max(`S`)` zwraca i usuwa maximum z S

Kolejki priorytetowe

Operacje:

- `Insert(`S`,`x`)` $S \leftarrow S \cup \{x\}$
- `Maximum(`S`)` zwraca maximium z S
- `Extract-Max(`S`)` zwraca i usuwa maximum z S

```
Heap-Maximum(A)
if heap-size[A] ≥ 1
  then return A[1]
  else error “Kopiec pusty”
```

Czas: $O(1)$.

Heap-Extract-Max

Heap-Extract-Max(A)
if *heap-size*[A]<1 then
  error “kopiec pusty”
$max \leftarrow A[1]$
$A[1] \leftarrow A[$*heap-size*$[A]]$
heap-size[A]$\leftarrow$*heap-size*[A]-1
Heapify(A,1)
return max

Czas: $O(\lg n)$.

Heap-Insert

`Heap-Insert(`A`,` key`)`
heap-size`[`A`]`$\leftarrow$*heap-size*`[`A`]`$+1$
$i \leftarrow$*heap-size*`[`A`]`
`while` $i > 1$ `and` $A[$`Parent(`i`)`$] < key$ `do`
    $A[i] \leftarrow A[$`Parent(`i`)`$]$
    $i \leftarrow$ `Parent(`i`)`
$A[i] \leftarrow key$

Czas: $O(\lg n)$

– “spacer po ścieżce od nowego liścia do korzenia”.

Quicksort

Dziel: Elementy $A[p \ldots r]$ rozdzielamy na mniejsze do $A[p \ldots q]$ i większe do $A[q+1 \ldots r]$.

Zwyciężaj: Rekurencyjnie sortujemy $A[p \ldots q]$ i $A[q+1 \ldots r]$.

Połącz: Nic nie trzeba robić!

Quicksort

Dziel: Elementy $A[p \dots r]$ rozdzielamy na mniejsze do $A[p \dots q]$ i większe do $A[q+1 \dots r]$.

Zwyciężaj: Rekurencyjnie sortujemy $A[p \dots q]$ i $A[q+1 \dots r]$.

Połącz: Nic nie trzeba robić!

```
Quicksort(A,p,r)
1 if p < r then
2    q ← Partition(A,p,r)
3    Quicksort(A,p,q)
4    Quicksort(A,q+1,r)
```

Wywołujemy: `Quicksort(`A`,`1`,`*length*`[`A`])`.

Partition

```
Partition(A,p,r)
x ← A[p]
i ← p − 1
j ← r + 1
while TRUE do
  repeat j ← j − 1
    until A[j] ≤ x
  repeat i ← i + 1
    until A[i] ≥ x
  if i < j then
    zamień A[i] ↔ A[j]
  else return j        ▷  j ≤ r − 1
```

Uwagi: Zawsze $i \leq r$ i $j \geq p$ oraz i i j mogą się wyminąć o ≤ 1 pozycję.

Czas: $O(n)$.

Czas Quicksort (intuicje)

Najgorszy przypadek: Każdy podział tworzy jeden przedział 1-elementowy. Wtedy:

$$\begin{aligned} T(n) &= T(n-1) + \Theta(n) \\ &= \sum_{k=1}^{n} \Theta(k) \\ &= \Theta\left(\sum_{k=1}^{n} k\right) \\ &= \Theta(n^2) \end{aligned}$$

Czas Quicksort (intuicje)

Najgorszy przypadek: Każdy podział tworzy jeden przedział 1-elementowy. Wtedy:

$$
\begin{aligned}
T(n) &= T(n-1) + \Theta(n) \\
&= \sum_{k=1}^{n} \Theta(k) \\
&= \Theta\left(\sum_{k=1}^{n} k\right) \\
&= \Theta(n^2)
\end{aligned}
$$

Najlepszy przypadek: Zawsze tworzone są 2 równe przedziały. Wtedy:

$$
\begin{aligned}
T(n) &= 2T(n/2) + \Theta(n) \\
&= \Theta(n \lg n)
\end{aligned}
$$

Randomized-Quicksort

Algorytm *probabilistyczny* (*randomized*) – przebieg zależy od wygenerowanych wartości losowych (zwykle – odporny na “złośliwe” dane).

Randomized-Quicksort

Algorytm *probabilistyczny* (*randomized*) – przebieg zależy od wygenerowanych wartości losowych (zwykle – odporny na "złośliwe" dane).

Podział:

Randomized-Partition(A,p,r)
1 $i \leftarrow$ Random(p,r) $\triangleright$ całkowita z $[p \ldots r]$
2 zamień $A[p] \leftrightarrow A[i]$
3 return Partition(A,p,r)

Randomized-Quicksort

Algorytm *probabilistyczny* (*randomized*) – przebieg zależy od wygenerowanych wartości losowych (zwykle – odporny na "złośliwe" dane).

Podział:

`Randomized-Partition`(A,p,r)
$i \leftarrow$ `Random`(p,r) $\quad \triangleright$ `całkowita z` $[p \ldots r]$
`zamień` $A[p] \leftrightarrow A[i]$
`return Partition`(A,p,r)

Sortowanie:

`Randomized-Quicksort`(A,p,r)
`if` $p < r$ `then`
    $q \leftarrow$ `Randomized-Partition`(A,p,r)
    `Randomized-Quicksort`(A,p,q)
    `Randomized-Quicksort`(A,$q+1$,r)

Analiza najgorszego przypadku

$$T(n) = \max_{1\le q\le n-1}(T(q)+T(n-q))+\Theta(n)$$

zgadujemy, że: $T(n) \le c\cdot n^2$, dla pewnej stałej c

$$T(n) \le c\cdot \max_{1\le q\le n-1}(q^2+(n-q)^2)+\Theta(n)$$

Analiza najgorszego przypadku

$$T(n) \quad = \quad \max_{1 \leq q \leq n-1}(T(q) + T(n-q)) + \Theta(n)$$

zgadujemy, że: $T(n) \leq c \cdot n^2$, dla pewnej stałej c

$$T(n) \quad \leq \quad c \cdot \max_{1 \leq q \leq n-1}(q^2 + (n-q)^2) + \Theta(n)$$

Druga pochodna $x^2 + (n-x)^2$ jest > 0, więc $q^2 + (n-q)^2$ ma maximum w przedziale $1 \leq q \leq n-1$ na jednym z końców.Stąd:

$$\max_{1 \leq q \leq n-1}(q^2 + (n-q)^2) \leq 1^2 + (n-1)^2 = n^2 - 2(n-1)$$

$$\begin{aligned} T(n) &\leq cn^2 - 2c(n-1) + \Theta(n) \\ &\leq cn^2 \end{aligned}$$

(dobieramy c t.że $2c(n-1) \geq \Theta(n)$)

Wniosek: $T(n) = O(n^2)$

Analiza średniego przypadku

(Zakładamy, że elementy są parami różne.)

- Niech $x = A[p]$ w momencie wywołania `Partition` w `Randomized-Partition`.
- Niech *rank*(x) – ranga x w $A[p \dots r]$ (t.j. pozycja x po posortowaniu $A[p \dots r]$). Niech $n = r - p + 1$.
- Ppb. tego że *rank*$(x) = i$, dla $i = 1, 2, \dots n$ wynosi $1/n$.
- Jeśli *rank*$(x) = 1$, to na końcu `Partition` $i = j = p$ (dolna część podziału zawiera 1 element). Ten przypadek zachodzi z ppb $1/n$
- Jeśli *rank*$(x) = i$, dla $i = 2, \dots, n$, to x trafi do do górnej części podziału, a każdy z *rank*$(x) - 1$ elementów mniejszych od x trafi do dolnej części. Rozmiar dolnej części: $i - 1$. Ppb każdego z tych przypadków: $1/n$.

Rekurencja

$$T(n) = \frac{1}{n}\left((T(1) + T(n-1)) + \sum_{q=1}^{n-1} (T(q) + T(n-q)) \right) + \Theta(n)$$

Z analizy najgorszego przypadku:
$T(1) = \Theta(1)$, $T(n-1) = O(n^2)$. Stąd:
$\frac{1}{n}(T(1) + T(n-1)) = \frac{1}{n}(\Theta(1) + O(n^2)) = O(n)$. (Można upchnąć w składnik $\Theta(n)$.)

$$\begin{aligned} T(n) &= \frac{1}{n}\left(\sum_{q=1}^{n-1}(T(q) + T(n-q))\right) + \Theta(n) \\ &= \frac{2}{n}\sum_{k=1}^{n-1} T(k) + \Theta(n) \end{aligned}$$

Rekurencja

$$T(n) = \frac{2}{n}\sum_{k=1}^{n-1} T(k) + \Theta(n)$$

Zał. ind.: $T(n) \leq an\lg n + b$ dla pewnych stałych $a, b > 0$

$$\begin{aligned} T(n) &\leq \frac{2}{n}\sum_{k=1}^{n-1}(ak\lg k + b) + \Theta(n) \\ &= \frac{2a}{n}\sum_{k=1}^{n-1}(k\cdot\lg k) + \frac{2b(n-1)}{n} + \Theta(n) \end{aligned}$$

Rekurencja

$$T(n) = \frac{2}{n}\sum_{k=1}^{n-1} T(k) + \Theta(n)$$

Zał. ind.: $T(n) \leq an\lg n + b$ dla pewnych stałych $a, b > 0$

$$\begin{aligned} T(n) &\leq \frac{2}{n}\sum_{k=1}^{n-1}(ak\lg k + b) + \Theta(n) \\ &= \frac{2a}{n}\sum_{k=1}^{n-1}(k \cdot \lg k) + \frac{2b(n-1)}{n} + \Theta(n) \end{aligned}$$

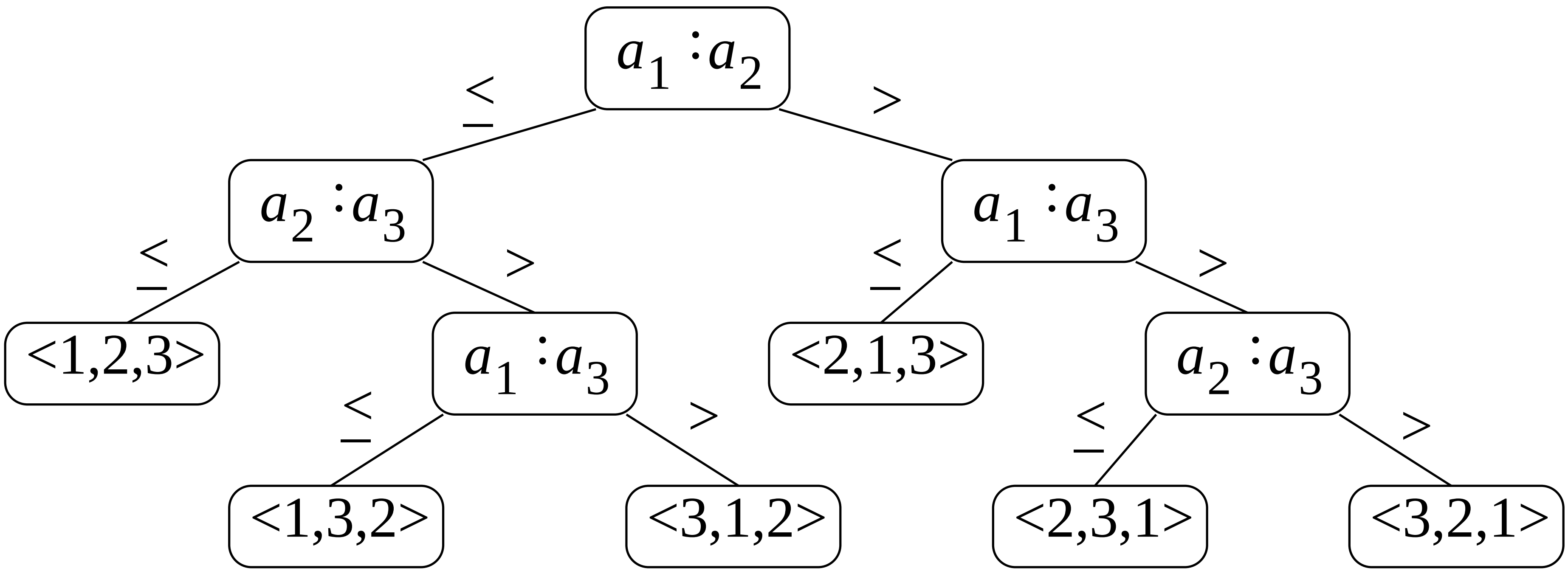
$$\begin{aligned} \sum_{k=1}^{n-1}(k \cdot \lg k) &= \sum_{k=1}^{\lceil n/2\rceil - 1}(k \cdot \lg k) + \sum_{k=\lceil n/2\rceil}^{n-1}(k \cdot \lg k) \\ &\quad \text{(korzystamy z: } \lg(n/2) = \lg n - 1\text{)} \\ &\leq (\lg n - 1)\sum_{k=1}^{\lceil n/2\rceil - 1} k + \lg n \sum_{k=\lceil n/2\rceil}^{n-1} k \\ &= \lg n \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lceil n/2\rceil - 1} k \\ &\leq \frac{1}{2}n(n-1)\lg n - \frac{1}{2}\left(\frac{n}{2} - 1\right)\frac{n}{2} \\ &\leq \frac{1}{2}n^2\lg n - \frac{1}{8}n^2 \qquad \text{dla } n \geq 2 \end{aligned}$$

Rekurencja

$$
\begin{aligned}
T(n) &\le \tfrac{2a}{n}\left(\tfrac{1}{2}n^2\lg n - \tfrac{1}{8}n^2\right) + \tfrac{2b(n-1)}{n} + \Theta(n) \\
&\le an\lg n - \tfrac{a}{4}n + 2b + \Theta(n) \\
&= an\lg n + b + \left(\Theta(n) + b - \tfrac{a}{4}n\right) \\
&\quad a \text{ dobrane tak, że } \tfrac{a}{4}n \ge \Theta(n) + b \\
&\le an\lg n + b
\end{aligned}
$$

Stąd średni czas `Randomized-Quicksort`:
$T(n) = O(n\lg n)$

Drzewo decyzyjne



drzewo dla `Insertion-Sort` na ciągu 3-elementowym:
$3! = 6$ liści.
Wykonanie – przejście od korzenia do liścia.

Konstrukcja drzewa decyzyjnego

Algorytm sortujący za pomocą porównań: Wyznacza wynik jedynie na podstawie wyników porównań. W każdej chwili: decyzje dotyczące dalszego przebiegu – jedynie na podstawie wyników dotychczasowych porównań.

Konstrukcja drzewa decyzyjnego

Algorytm sortujący za pomocą porównań: Wyznacza wynik jedynie na podstawie wyników porównań. W każdej chwili: decyzje dotyczące dalszego przebiegu – jedynie na podstawie wyników dotychczasowych porównań.
Konstrukcja drzewa dla danego algorytmu i rozmiaru danych (przy założeniu, że elementy są parami różne):

- korzeń – pierwsze porównanie
- lewe (odp. prawe) podrzewo – drzewo dla pozostałej części algorytmu, jeśli wynikiem pierwszego porównania jest “$\leq$” (odp. “$>$”)
- liść – ustalona przez algorytm permutacja elementów początkowego ciągu, która daje ciąg posortowany

Dolna granica dla sortowania

Tw. Każde drzewo decyzyjne dla algorytmu sortującego n elementów ma wysokość $\Omega(n \lg n)$.

Dowód. Drzewo musi mieć $\geq n!$ liści. (Każde początkowe uporządkowanie jest możliwe.)
Niech h wysokość drzewa. Zatem liści jest $\leq 2^h$.
Stąd: $n! \leq 2^h$, czyli $\lg(n!) \leq h$.
Ze wzoru Stirlinga:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right) \geq \left(\frac{n}{e}\right)^n$$

Stąd: $h \geq \lg\left(\frac{n}{e}\right)^n = n \lg n - n \lg e = \Omega(n \lg n)$ $\square$

Dolna granica dla sortowania

Tw. Każde drzewo decyzyjne dla algorytmu sortującego n elementów ma wysokość $\Omega(n \lg n)$.

Dowód. Drzewo musi mieć $\geq n!$ liści. (Każde początkowe uporządkowanie jest możliwe.)

Niech h wysokość drzewa. Zatem liści jest $\leq 2^h$.

Stąd: $n! \leq 2^h$, czyli $\lg(n!) \leq h$.

Ze wzoru Stirlinga:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right) \geq \left(\frac{n}{e}\right)^n$$

Stąd: $h \geq \lg\left(\frac{n}{e}\right)^n = n \lg n - n \lg e = \Omega(n \lg n)$ $\square$

Wniosek. `Merge-Sort` i `Heapsort` są asymptotycznie optymalne

Counting-Sort

Dane: $A[1 \ldots n]$ ciąg liczb całk. z przedz. $1, \ldots, k$, $k = O(n)$.

Counting-Sort(A,B,k)
for $i \leftarrow 1$ to k do
    $C[i] \leftarrow 0$
for $j \leftarrow 1$ to *length*[A] do
    $C[A[j]] \leftarrow C[A[j]] + 1$
$\triangleright$ $C[i] =$ (liczba elementów i)
for $i \leftarrow 2$ to k do
    $C[i] \leftarrow C[i] + C[i-1]$
$\triangleright$ $C[i] =$ (liczba elementów $\leq i$)
for $j \leftarrow$*length*[A] downto 1 do
    $B[C[A[j]]] \leftarrow A[j]$
    $C[A[j]] \leftarrow C[A[j]] - 1$

Czas: $O(n+k)$. Dla $k = O(n)$: $O(n)$.

Counting-Sort – uwagi

- wynik w $B[1 \ldots n]$
- $C[1 \ldots k]$ - tablica pomocnicza
- nie jest to algorytm sortujący za pomocą porównań
- jest *stabilny* (tzn. liczby o tych samych wartościach pozostają w tej samej kolejności co były na początku)

Radix-Sort

Dane: $A[1 \ldots n]$ ciąg liczb d-cyfrowych w systemie o podstawie k (cyfra najmniej znacząca – na pozycji 1).

```
Radix-Sort(A,d)
1 for i ← 1 to d do
2   posortuj stabilnie A wg cyfry i
```

Jeśli w linii 2 stosujemy `Counting-Sort`, to czas: $\Theta(dn + kd)$.

Np. sortowanie liczb 64-bitowych, traktowanych jako czterocyforwe w systemie o podst. 2^{16} wymaga 4 przebiegów `Counting-Sort`.

Bucket-Sort

Dane: liczby losowe z przedz. $[0, 1)$ wybierane z rozkładem jednostajnym.

Bucket-Sort(A)
$n \leftarrow$ *length*[A]
for $i \leftarrow 1$ to n do
  wstaw $A[i]$ do listy $B[\lfloor nA[i] \rfloor]$
for $i \leftarrow 0$ to $n-1$ do
  posortuj listę $B[i]$
połącz listy $B[0], \ldots, B[n-1]$ w tej kolejności

Bucket-Sort

Dane: liczby losowe z przedz. $[0,1)$ wybierane z rozkładem jednostajnym.

```
Bucket-Sort(A)
1 n ← length[A]
2 for i ← 1 to n do
3   wstaw A[i] do listy B[⌊nA[i]⌋]
4 for i ← 0 to n − 1 do
5   posortuj listę B[i]
6 połącz listy B[0], ..., B[n − 1] w tej kolejności
```

Uwaga: w wierszu 5 może być `Insertion-sort` dla listy.

Niech n_i – zmienna losowa – liczba el. w $B[i]$. Oczekiwany czas posortowania $B[i]$: $E[O(n_i^2)] = O(E[n_i^2])$.

Oczekiwany czas wszystkich sortowań w wierszu 5:

$\sum_{i=0}^{n-1} O(E[n_i^2]) = O\left(\sum_{i=0}^{n-1} E[n_i^2]\right)$.

Rozkład dwumianowy

p – ppb. sukcesu, $q = 1 - p$ – ppb. porażki.

$b(k; n, p) = \binom{n}{k} p^k (1-p)^{n-k}$ – ppb. k sukcesów w n próbach.

X – zm. losowa – liczba sukcesów w n *niezależnych* próbach

$X_i \in \{0, 1\}$ – zm. losowa – liczba sukcesów w i-tej próbie.

Wartość oczekwana:

$E[X_i] = p \cdot 1 + q \cdot 0 = p$.

$E[X] = E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} p = np$

Wariancja:

$Var[X_i] = E[X_i^2] - E^2[X_i]$.

Ponieważ zawsze $X_i = X_i^2$, mamy: $E[X_i^2] = E[X_i] = p$.

Stąd: $Var[X_i] = p - p^2 = pq$.

Z *niezależności* prób:

$Var[X] = Var\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} Var[X_i] = \sum_{i=1}^{n} pq = npq$.

Bucket-Sort – c.d.

Ppb, że $n_i = k$ wynosi $b(k; n, p)$, gdzie $p = 1/n$. Stąd $E[n_i] = np = 1$ oraz $Var[n_i] = np(1-p) = 1 - 1/n$. Z rowności $Var[X] = E[X^2] - E^2[X]$ mamy:

$$\begin{aligned} E[n_i^2] &= Var[n_i] + E^2[n_i] \\ &= 1 - \frac{1}{n} + 1^2 \\ &= 2 - \frac{1}{n} \\ &= \Theta(1) \end{aligned}$$

Stąd oczekiwany czas `Bucket-Sort`:
$O\left(\sum_{i=0}^{n-1} E[n_i^2]\right) = O(n)$. (Łączny czas spędzony poza wierszem 5 wynosi również $O(n)$.)

Mediany i statystyki pozycyjne

i*-ta statystyka pozycyjna* – i-ty co do wielkości element w zbiorze n-elementowym. (Np. *minimum* dla $i = 1$ oraz *maksimum* dla $i = n$.)

Mediana – element "środkowy". (Dla n parzystego – dwie mediany.)

Problem wyboru:

Dane: Zbiór A (parami różnych) n liczb, oraz i, $1 \leq i \leq n$.

Wynik: Element $x \in A$ większy od dokładnie $i - 1$ elementów A.

Randomized-Select

Wybieramy i-ty co do wielkości element w $A[p \ldots r]$.

```
Randomized-Select(A, p, r, i)
if p = r then
    return A[p]
q ← Randomized-Partition(A, p, r)
k ← q − p + 1
if i ≤ k then
    return Randomized-Select(A, p, q, i)
else
    return Randomized-Select(A, q + 1, r, i − k)
```

Randomized-Select

Wybieramy i-ty co do wielkości element w $A[p \ldots r]$.

```
Randomized-Select(A,p,r,i)
if p = r then
  return A[p]
q ← Randomized-Partition(A,p,r)
k ← q − p + 1
if i ≤ k then
  return Randomized-Select(A,p,q,i)
else
  return Randomized-Select(A,q+1,r,i−k)
```

Najgorszy przypadek – czas: $\Theta(n^2)$ (Np. szukamy minimum i `Randomized-Partition` zawsze dzieli wzgl. największego elementu.)

Randomized-Select – czas oczekiwany

W `Randomized-Partition` dolna część podziału ma rozmiar 1 z ppb $2/n$ oraz i z ppb. $1/n$, dla $i = 1, \ldots, n-1$. (Zakł. że: $T(n)$ rosnąca.) W najgorszym przyp. i-ty element zawsze jest w większej części podziału.

$$
\begin{aligned}
T(n) &\leq \frac{1}{n}\left(T(\max(1, n-1)) + \sum_{k=1}^{n-1} T(\max(k, n-k))\right) + O(n) \\
&\leq \frac{1}{n}\left(T(n-1) + 2\sum_{k=\lceil n/2 \rceil}^{n-1} T(k)\right) + O(n)
\end{aligned}
$$

(korzystamy z: $\frac{1}{n}T(n-1) = \frac{1}{n}O(n^2) = O(n)$)

$$
\leq \frac{2}{n}\sum_{k=\lceil n/2 \rceil}^{n-1} T(k) + O(n)
$$

Randomized-Select – czas oczekiwany

Zał. ind.: $T(n) \le cn$ dla pewnej stałej c.

$$
\begin{aligned}
T(n) &\le \frac{2}{n}\sum_{k=\lceil n/2\rceil}^{n-1} ck + O(n) \\
&\le \frac{2c}{n}\left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lceil n/2\rceil -1} k\right) + O(n) \\
&= \frac{2c}{n}\left(\frac{1}{2}(n-1)n - \frac{1}{2}(\lceil n/2\rceil - 1)\lceil n/2\rceil\right) + O(n) \\
&\le c(n-1) - \frac{c}{n}(n/2-1)(n/2) + O(n) \\
&= c\left(\frac{3}{4}n - \frac{1}{2}\right) + O(n) \\
&\quad (c \text{ dobrane tak, że: } c(n/4+1/2) \ge O(n)) \\
&\le cn
\end{aligned}
$$

Stąd – czas oczekiwany: $O(n)$.

Select

Select(i)

1. Podziel n elementów na $\lfloor n/5 \rfloor$ grup 5-elementowych i ≤ 1 ($n \bmod 5$)-elementową
2. Wyznacz medianę w każdej z $\lceil n/5 \rceil$ grup. (Jeśli ostatnia grupa - parzysta, to - większą z jej median).
3. Wywołaj rekurencyjnie Select aby wyznaczyć: x - medianę z $\lceil n/5 \rceil$ median.
4. Podziel elementy wzgl. x (podobnie jak Partition). Niech k - liczność dolnej części podziału.
5. Jeśli $i \leq k$ to rekurencyjnie - Select(i) w dolnej części podziału. W p.p. - Select($i - k$) w górnej części.

Select – analiza

(Zał.: elementy parami różne.) Liczba el. większych od x jest $\geq 3(\lceil 1/2\lceil n/5\rceil\rceil - 2) \geq 3n/10 - 6$. (W pełnych grupach o medianach $> x$ są po 3 el. większe od x. Z $\lceil 1/2\lceil n/5\rceil\rceil$ grup o medianach $\geq x$ pomijamy dwie: niepełną i zawierającą x.)

Select – analiza

(Zał.: elementy parami różne.) Liczba el. większych od x jest $\geq 3(\lceil 1/2 \lceil n/5 \rceil \rceil - 2) \geq 3n/10 - 6$. (W pełnych grupach o medianach $> x$ są po 3 el. większe od x. Z $\lceil 1/2 \lceil n/5 \rceil \rceil$ grup o medianach $\geq x$ pomijamy dwie: niepełną i zawierającą x.) Analogicznie liczba el. $< x$ jest $\geq 3n/10 - 6$. W najgorszym razie `Select` jest wywołane w ostatnim kroku dla zbioru $n - (3n/10 - 6) = (7n/10 + 6)$-elementowego.

Select – analiza

(Zał.: elementy parami różne.) Liczba el. większych od x jest $\geq 3(\lceil 1/2 \lceil n/5 \rceil \rceil - 2) \geq 3n/10 - 6$. (W pełnych grupach o medianach $> x$ są po 3 el. większe od x. Z $\lceil 1/2 \lceil n/5 \rceil \rceil$ grup o medianach $\geq x$ pomijamy dwie: niepełną i zawierającą x.) Analogicznie liczba el. $< x$ jest $\geq 3n/10 - 6$. W najgorszym razie `Select` jest wywołane w ostatnim kroku dla zbioru $n - (3n/10 - 6) = (7n/10 + 6)$-elementowego.
Niech $T(n)$ pesymistyczny czas dla n elementów. Kroki 1,2 i 4 – czas: $O(n)$. Krok 3: $T(\lceil n/5 \rceil)$. Krok 5: $T(7n/10 + 6)$.

Select – analiza

(Zał.: elementy parami różne.) Liczba el. większych od x jest $\geq 3(\lceil 1/2\lceil n/5\rceil\rceil - 2) \geq 3n/10 - 6$. (W pełnych grupach o medianach $> x$ są po 3 el. większe od x. Z $\lceil 1/2\lceil n/5\rceil\rceil$ grup o medianach $\geq x$ pomijamy dwie: niepełną i zawierającą x.) Analogicznie liczba el. $< x$ jest $\geq 3n/10 - 6$. W najgorszym razie `Select` jest wywołane w ostatnim kroku dla zbioru $n - (3n/10 - 6) = (7n/10 + 6)$-elementowego.
Niech $T(n)$ pesymistyczny czas dla n elementów. Kroki 1,2 i 4 – czas: $O(n)$. Krok 3: $T(\lceil n/5\rceil)$. Krok 5: $T(7n/10 + 6)$.
Uwagi: $7n/10 + 6 < n$ dla $n > 20$ oraz dla $n < 80$ – czas: $O(1)$.

Select – analiza

$$T(n) \leq \begin{cases} \Theta(1), & \text{dla } n \leq 80 \\ T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n), & \text{dla } n > 80 \end{cases}$$

Select – analiza

$$T(n) \leq \begin{cases} \Theta(1), & \text{dla } n \leq 80 \\ T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n), & \text{dla } n > 80 \end{cases}$$

Zał. ind.: $T(n) \leq cn$ dla pewnej stałej c.

$$\begin{aligned} T(n) &\leq c\lceil n/5 \rceil + c(7n/10 + 6) + O(n) \\ &\leq cn/5 + c + 7cn/10 + 6c + O(n) \\ &\leq 9cn/10 + 7c + O(n) \\ &\leq cn \quad (c \text{ dobrane tak, że } c(n/10 - 7) > O(n)) \end{aligned}$$

Wniosek: pesymistyczny czas `Select`: $O(n)$.