Fakultas Informatika
School of Computing
Telkom University
Think Big! Go Forward!

Telkom University

ANALISIS ALGORITMA

Week 03: Strategi Algoritma

PROGRAM PASCA SARJANA INFORMATIKA
FAKULTAS TEKNIK INFORMATIKA
UNIVERSITAS TELKOM

2022/2023

Strategi Algoritma

bagian 2 dari 3

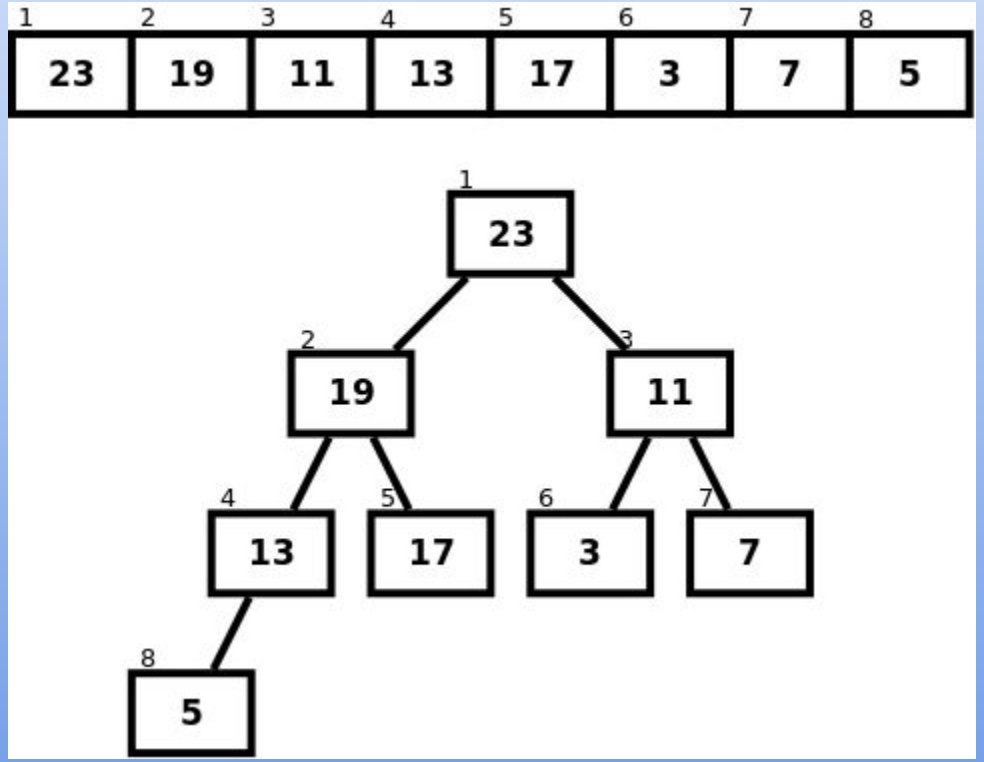
- Kasus Sorting Heapsort

HeapSort
Penyempurnaan
Selection sort

Heap prioritas

- FindMax sangat efisien untuk satu kali operasi
  - (karena semua data -kan harus diperiksa)
- Tetapi jika dilakukan berulang-kali, pengulangan proses mungkin terjadi
  - Perlu metoda lain untuk menghindari hal tersebut
- Heap adalah pohon biner dengan properti berikut:
  - Nilai tersimpan setiap node tidak lebih kecil dari nilai kedua anak-nya
  - Dapat disimpulkan, root pastilah mempunyai nilai terbesar
  - Semua daun, yaitu node tanpa anak, otomatis memenuhi syarat tersebut
  - **Perhatikan** tidak ada syarat bahwa anak kiri harus lebih kecil/besar dari anak yang kanan

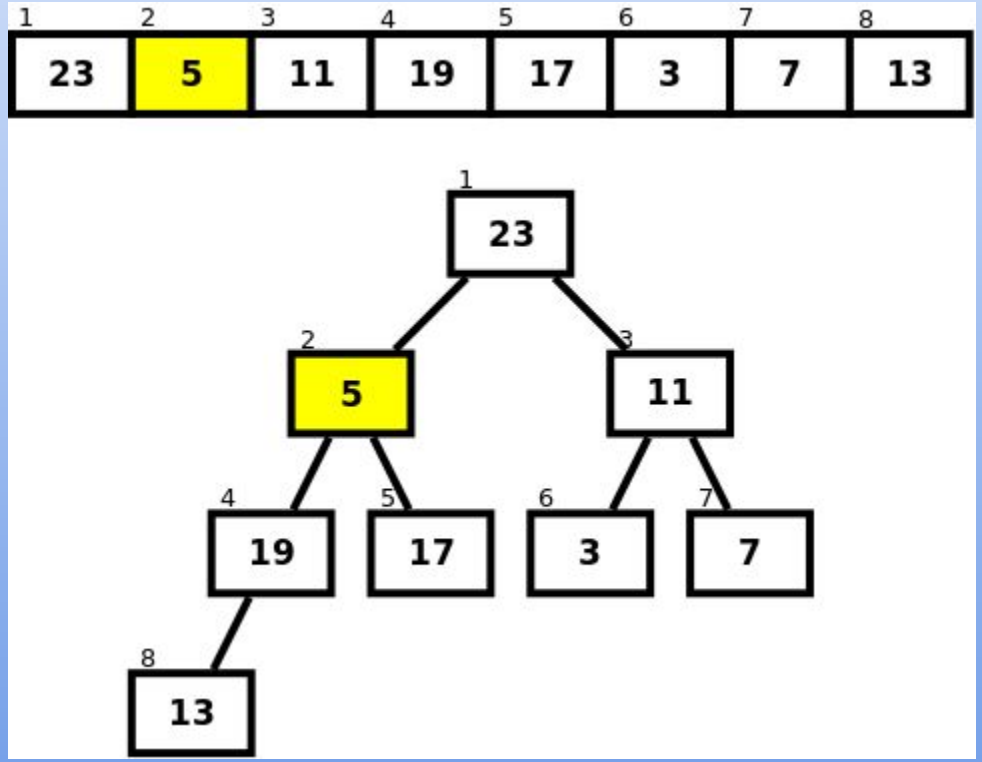
Struktur Heap dapat disimpan dalam Array



Satu Node Salah Posisi ⇓

- Keseluruhan heap hampir benar, kecuali satu node
- Nilai node tersebut (mungkin) lebih kecil dari salah satu atau kedua anaknya
- Lakukan operasi swap dengan anak-nya
- Tapi mungkin perlu propagasi, jika setelah swap, nilainya juga masih lebih kecil dari nilai cucu-nya

Satu Node Salah Posisi ⇓



Perbaiki Heap ⇓

```
{ p adalah indeks ke satu²nya node yang melanggar properti heap,
   nilai pada arr[p] lebih kecil dari nilai (salah satu) anaknya }
proc FixHeap( arr[1..n], p )
    bigger := p
    more2fix := true
    while more2fix do
        more2fix := false
        kr := p*2
        kn := kr + 1
        if kr ≤ n and arr[kr] > arr[bigger] then bigger := kr
        if kn ≤ n and arr[kn] > arr[bigger] then bigger := kn
        if p <> bigger then
            arr[p] ⇄ arr[bigger]
            p := bigger
            more2fix := true
        endif
    endwhile
endproc
```

Satu Node Salah Posisi ⇓

- **Hanya** node arr[p] yang mungkin melanggar properti heap
- arr[1..p-1] dan arr[p+1]..n], 1 ≤ p ≤ n, memenuhi properti heap
- Setelah satu iterasi:
  - Properti node p diperbaiki
  - Mungkin node 2p atau 2p+1 melanggar properti
  - Invarian tetap berlaku (tidak lebih dari satu node yang melanggar properti)
- Saat terminasi:
  - Properti node p tidak melanggar, atau
  - p adalah leaf (2p dan 2p+1 lebih besar dari n)
  - Maka heap kembali benar untuk semua node
- **Bagaimana jika ada lebih dari satu node yang melanggar properti heap?**

Satu Node Salah Posisi ⇓

- Kompleksitas bergantung pada jumlah iterasi (s.d. more2fix menjadi false)
- Setiap iterasi nilai p berubah menjadi 2p atau 2p+1,
  - sampai dengan p adalah leaf, atau
  - arr[p] bukan node yang salah lagi
- Jumlah iterasi adalah jarak dari p awal s.d. leaf dalam pohon heap
  - yaitu sebanyak . . .

Pohon Biner, Leaf, Tinggi(h), Kedalaman(d)

- Pohon biner lengkap (CBT) merupakan pohon biner yang hampir sempurna
  - semua tingkatan berisi node
  - pada tingkat paling bawah, node mengisi sayap kiri lebih dulu (lihat gambar sebelumnya)
- Jika CBT mempunyai n node maka
  - Jumlah leaf = $\lceil \frac{1}{2}n \rceil$ **(Buktikan dengan menggunakan induksi)**
  - Tinggi node p adalah jarak/jumlah edge dari node p ke leaf, **h(p)**, p=1,2..n
  - Tinggi pohon adalah jarak/jumlah edge dari root ke leaf terbawah, yaitu $\mathbf{h = \lfloor lg(n) \rfloor}$
  - Kedalaman suatu node p adalah jarak/jumlah edge dari root ke p, yaitu $\mathbf{d(p) = \lfloor lg(p) \rfloor}$
  - Jumlah node pada tinggi **i** ada sebanyak $\mathbf{n^i}$, dimana **i=0..h-1** (kecuali level terbawah)
  - (∴)Maka tinggi node p adalah tinggi pohon dikurang kedalaman p
    - $\mathbf{n \geq 2^{h(p)}p}$ maka **h(p) = h - d(p)**
    - $\mathbf{n < 2^{h(p)}p}$ maka **h(p) = h - d(p) - 1**

Satu Node Salah Posisi ⇓

- Kompleksitas bergantung pada jumlah iterasi (s.d. **more2fix** menjadi **false**)
- Setiap iterasi nilai **p** berubah menjadi **2p** atau **2p+1**,
  - sampai dengan **p** adalah leaf, atau
  - **arr[p]** bukan node yang salah lagi
- Maksimum jumlah iterasi adalah jarak dari **p** awal s.d. leaf dalam pohon heap
  - yaitu sebanyak tinggi p, atau T(n,p) = h - d(p) = ⌊*lg*(n)⌋ - ⌊*lg*(p)⌋
  - Kasus terburuk terjadi jika node salah posisi adalah root, sehingga T(n,p)=**T(n,1)=⌊*lg*(n)⌋**
  - Kasus terbaik terjadi ketika node salah posisi adalah parent dari leaf, atau ¼n ≤ p < ½n, sehingga **T(n,p) = 1**
  - Kasus rerata . . .

Satu Node Salah Posisi ⇓

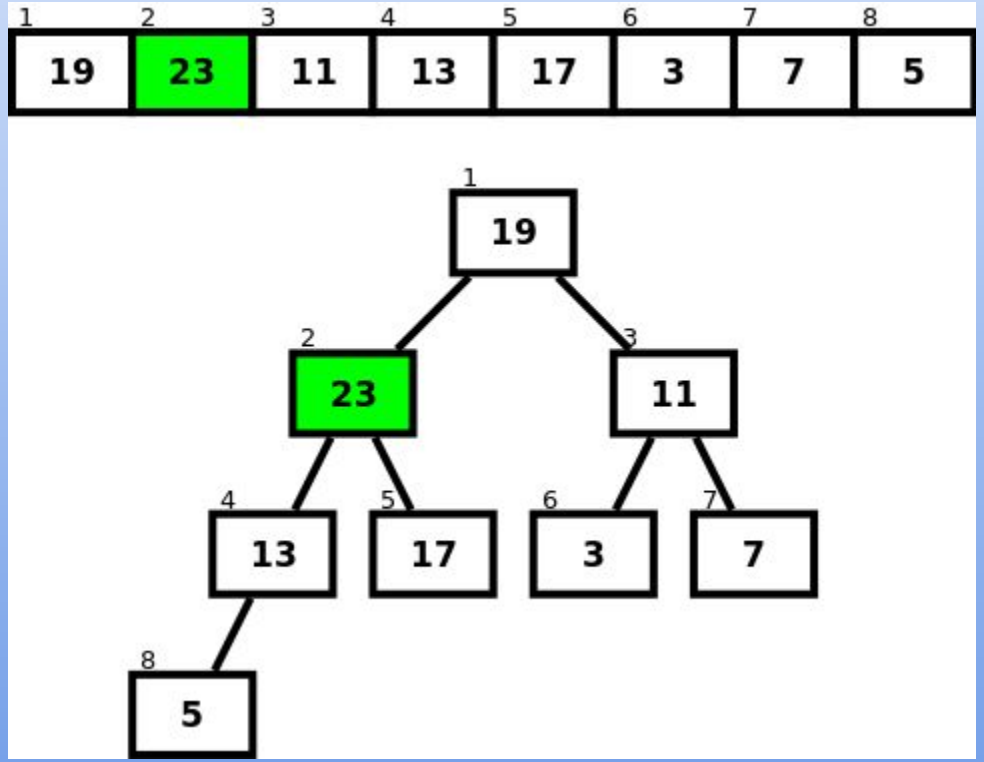
- Kompleksitas bergantung pada jumlah iterasi (s.d. more2fix menjadi false)
- Setiap iterasi nilai p berubah menjadi 2p atau 2p+1,
  - sampai dengan p adalah leaf, atau
  - arr[p] bukan node yang salah lagi
- Maksimum jumlah iterasi adalah jarak dari p awal s.d. leaf dalam pohon heap
  - yaitu sebanyak tinggi p, atau T(n,p) = h - d(p) = ⌊*lg*(n)⌋ - ⌊*lg*(p)⌋
  - Kasus terburuk terjadi jika node salah posisi adalah root, sehingga T(n,p)=**T(n,1)=⌊*lg*(n)⌋**
  - Kasus terbaik terjadi ketika node salah posisi adalah parent dari leaf, atau ¼n ≤ p < ½n, sehingga **T(n,p) = 1**
  - Kasus rerata dengan asumsi node p mungkin node mana saja diluar leaf adalah **T(n)** = (¼n + ¹/₈n×2 + ¹/₁₆n×3 + ¹/₃₂n×4 + … + 1×*lg*(n/2))/½n ≃n×*lg*(n) /½n ≃ 2*lg*(n)

Satu Node Salah Posisi ⇓

Kasus rerata dengan asumsi node p mungkin node mana saja diluar leaf

- Jumlah data = ½n node
- Biaya per node = $T_n(n,p) = \lfloor lg(n) \rfloor - \lfloor lg(p) \rfloor$, $p=1..\lfloor ½n \rfloor$
- Biaya per depth = $T_d(n,d) = 2^d.T(n,p=2^d)=2^d(\lfloor lg(n) \rfloor - d)$, $d=0..\lfloor lg(n) \rfloor-1$
- $\sum_{d=0..\lfloor lg(n) \rfloor-1} 2^d = 1+2+4+..+½n = n$
- $\sum_{d=0..\lfloor lg(n) \rfloor-1} 2^d d = 0+2+8+24+..(½n.lgn-½n) <= ¾n.lg(n)$
- Total biaya = $T_{tot}(n) = \sum_{d=0..\lfloor lg(n) \rfloor-1} T(n,d) = n lg(n) - ¾n.lg(n) = ¼n.lg(n)$
- Rerata biaya = $T(n) = T_{tot}(n)/(½n) = ¼n.lg(n)/(½n)=½lg(n)$
- ½n+¼n.2+⅛n.3+...2.(lgn-1) + 1.lgn

Satu Node Salah Posisi ⇑



Perbaiki Heap ⇑

```
{ p pointer/indeks ke satu²nya node yang melanggar properti heap
  nilai pada arr[p] lebih besar dari nilai parent-nya }
proc UpFixHeap( arr[1..n], p )

     { silakan dibuat algoritmanya }

endproc
```

- Buat algoritma versi upfix
- Buktikan kebenaran algoritma ini
- Hitung kompleksitasnya

Bangun Heap

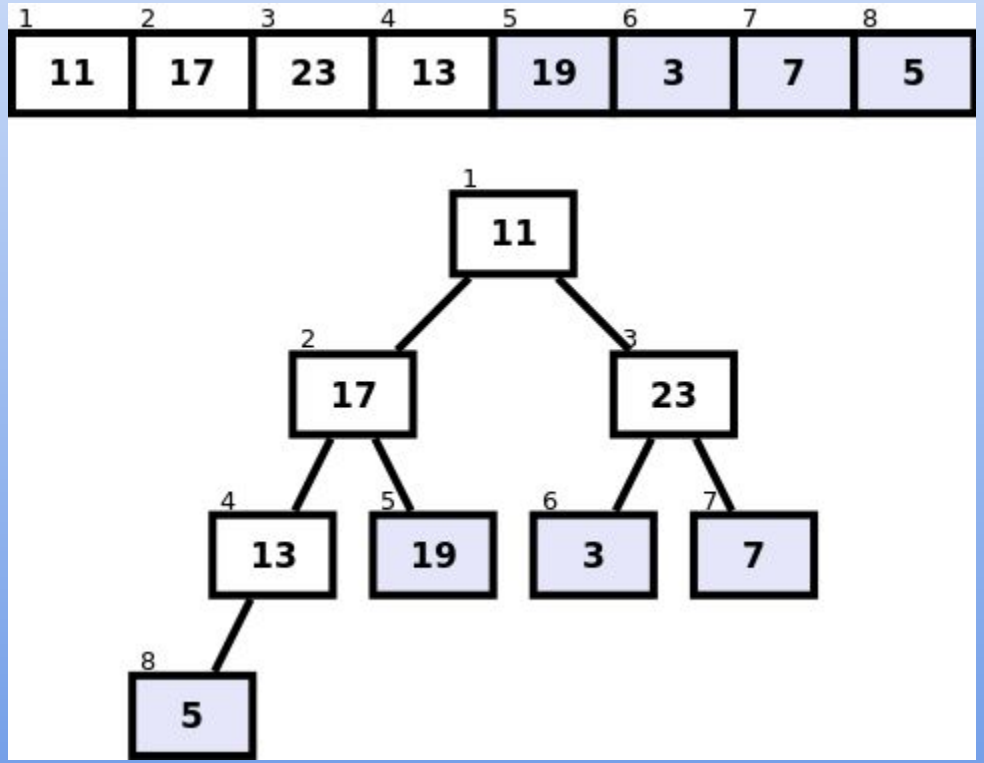
- Diberikan sejumlah data dalam array (dan belum membentuk heap)
- Susun data dalam array tersebut agar memenuhi properti heap!
- Perbaiki heap hanya berlaku jika yang tidak memenuhi properti heap hanya 1 node saja.

Bangun Heap

- Diberikan sejumlah data dalam array (dan belum membentuk heap)
- Susun data dalam array tersebut agar memenuhi properti heap!

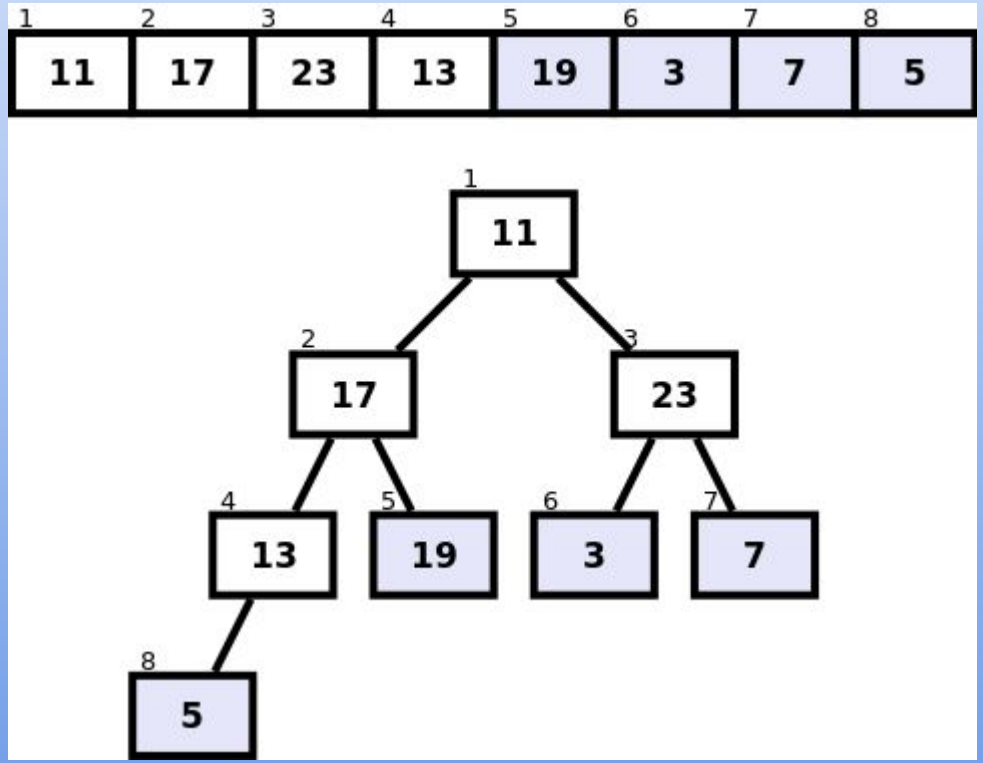
- Keadaan awal:
  - Semua leaf memenuhi properti heap
  - Ada sebanyak ⌈½n⌉ leaf dalam pohon biner dengan n node
- Selanjutnya, tambahkan satu persatu node tersisa menggunakan algoritma perbaiki heap

Bangun Heap



Bangun Heap

```
proc ConstructHeap( arr[1..n] )
     i := ⌊½n⌋
     while i ≥ 1 do
          FixHeap(arr[1..n], i)
          i := i - 1
     endwhile
endproc
```

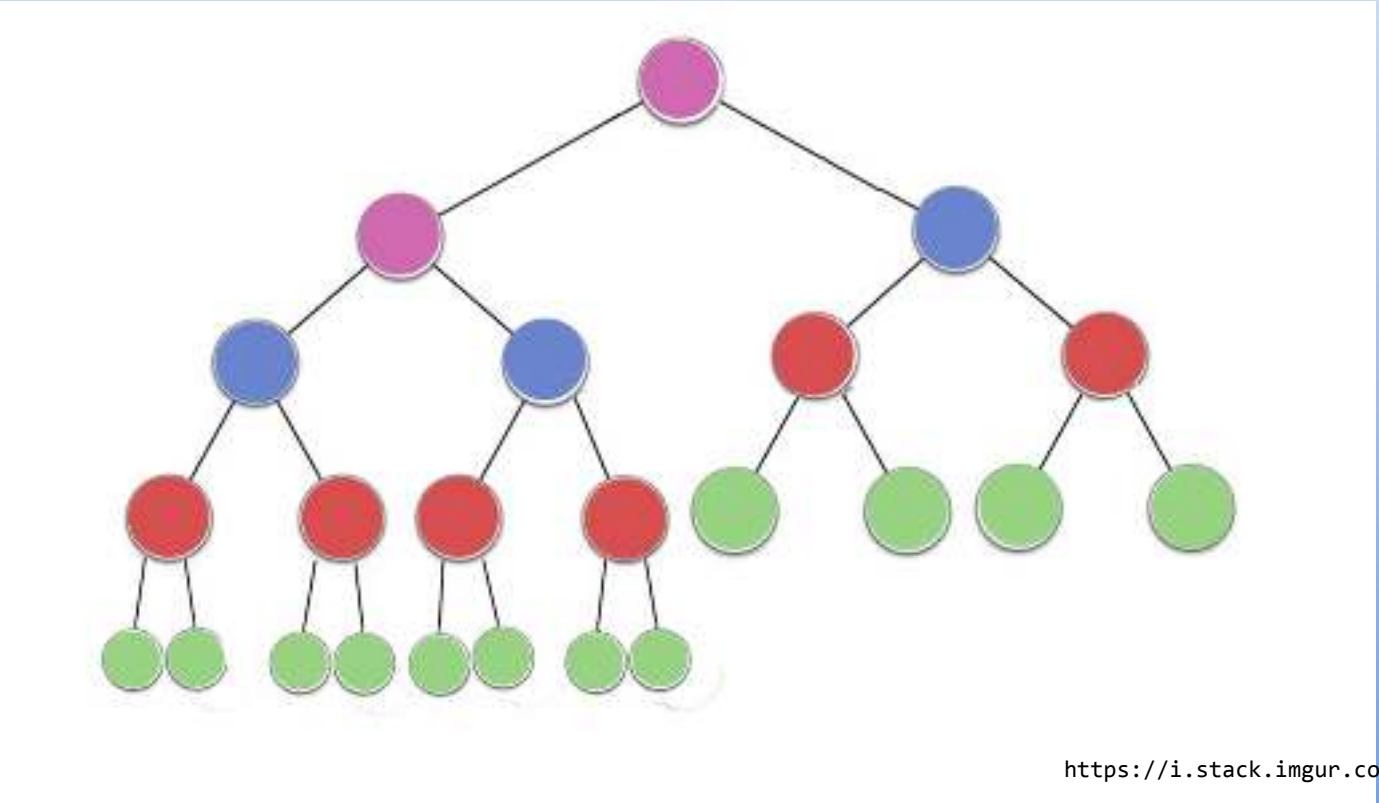


Bangun Heap

```
proc ConstructHeap( arr[1..n] )
     i := ⌊½n⌋
     while i ≥ 1 do
          FixHeap(arr[1..n], i)
          i := i - 1
     endwhile
endproc
```

- Bukti kebenaran?
  - Gunakan metoda induksi
- Perhitungan kompleksitas?
  - $T_{CONSTRUCTHEAP}(n)=O(n\lg n)$
  - Bisa lebih ketat lagi ?..
  - Bdk perhitungan rerata perbaikan heap

Bangun Heap



https://i.stack.imgur.com/rD6R0.jpg

Bangun Heap

- Tinggi pohon = h = $\log_2 n$
- Jumlah maksimum elemen pada tinggi t = $el_t = 2^{h-t}$
- Maks. biaya memperbaiki heap pada ketinggian 1 (parent leaf) = $c_1 = 1*2^{h-1}$
- Pada ketinggian 2 = $c_2 = 2*2^{h-2}$ dst. Shg biaya pada tinggi t = $c_t = t*2^{h-t}$
- Total biaya membangun heap = $T(n) = \sum_{t=1..h} c_t = t*2^{h-t} = 1*2^{h-1} + 2*2^{h-2} + ... + h*2^0$
- $T(n) = 2^h \left(1/2 + 2(1/2)^2 + 3(1/2)^3 + ... \quad h(1/2)^h\right)$
- Half T = $H(n) = T(n)/2 = 2^h \left((1/2)^2 + 2(1/2)^3 + 3(1/2)^4 + ... (h-1)(1/2)^h + h(1/2)^{h+1}\right)$
- $H'(n) = T(n) - H(n) = 2^h \left(1/2 + (1/2)^2 + (1/2)^3 + ... \quad (1/2)^h + h(1/2)^{h+1}\right)$
- $T(n) = 2H'(n) = 2^{h+1}(1/2 + 1/4 + 1/8 + 1/16 + ...) < 2^{h+1} * 1 = 2^{(\log n)+1} = 2n$
- Total biaya membangun heap = $T(n) = 2n = O(n)$

Heap Sort, Modifikasi Selection Sort

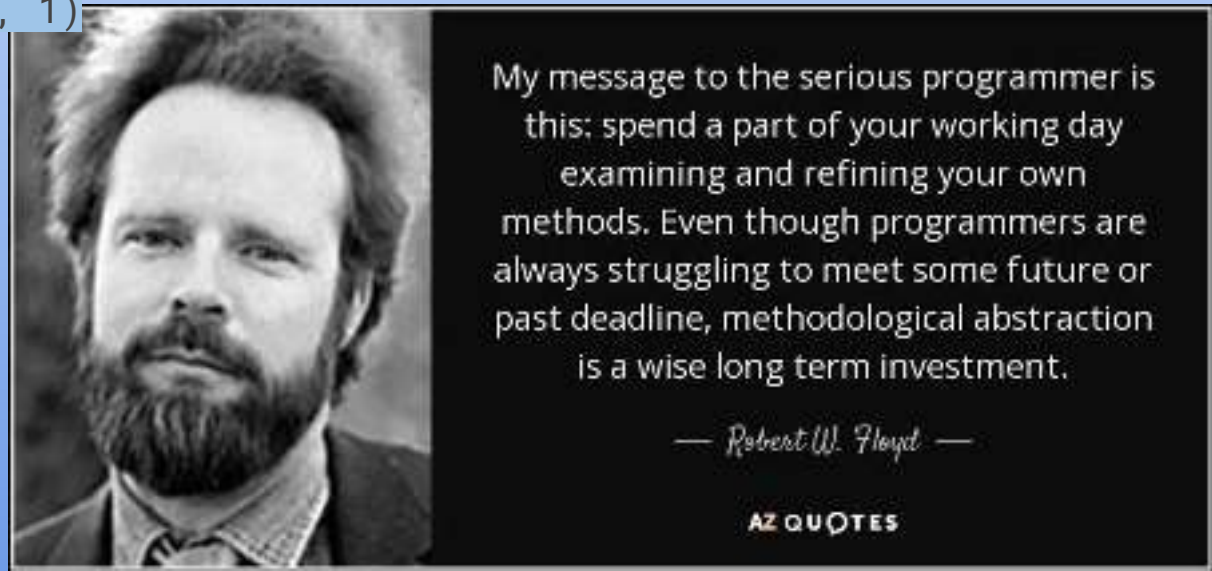
- FindMax diganti dengan struktur heap
- Nilai maksimum adalah root dari heap
- Perlu proses perbaikan heap setiap kali nilai maksimum diambil dari heap
- Perlu proses awal untuk membangun heap

Heap Sort

```
proc HeapSort( arr[1..n] )
    ConstrutHeap(arr[1..n])
    i := n
    while i > 1 do
        arr[i] ⇄ arr[1]
        FixHeap(arr[1..i-1], 1)
        i := i - 1
    endwhile
```

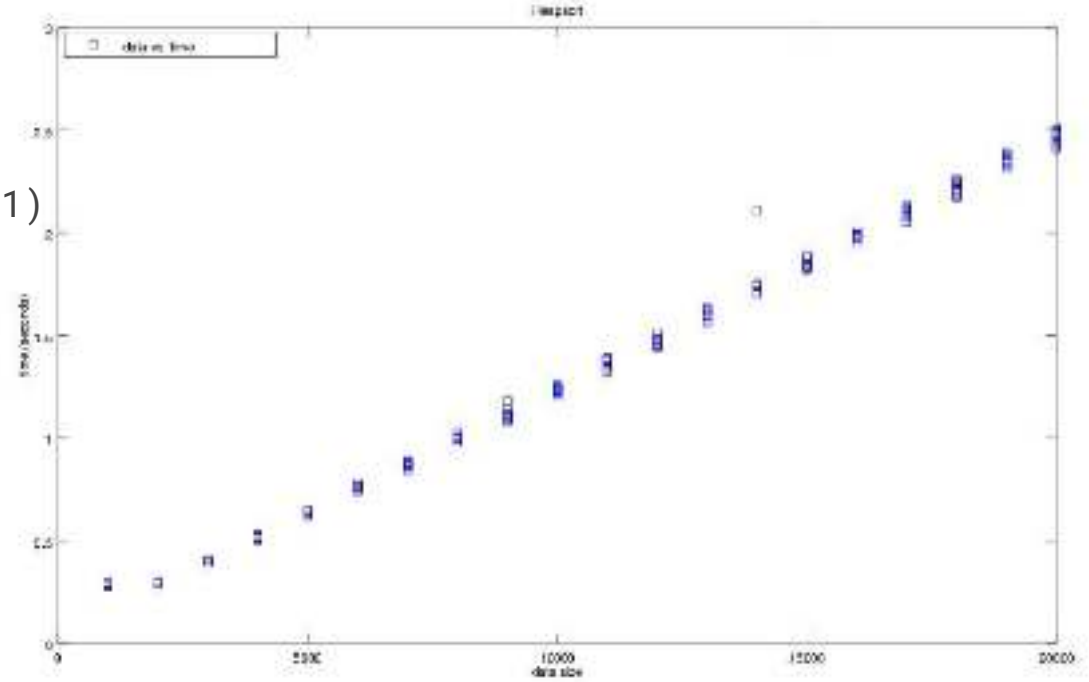
Diciptakan oleh:

- R.W. Floyd (1964) Treesort
- J.W.J Williams(1964) Heapsort



Heap Sort

```
proc HeapSort( arr[1..n] )
    ConstructHeap(arr[1..n])
    i := n
    while i > 1 do
        arr[i] ⇄ arr[1]
        FixHeap(arr[1..i-1], 1)
        i := i - 1
    endwhile
```

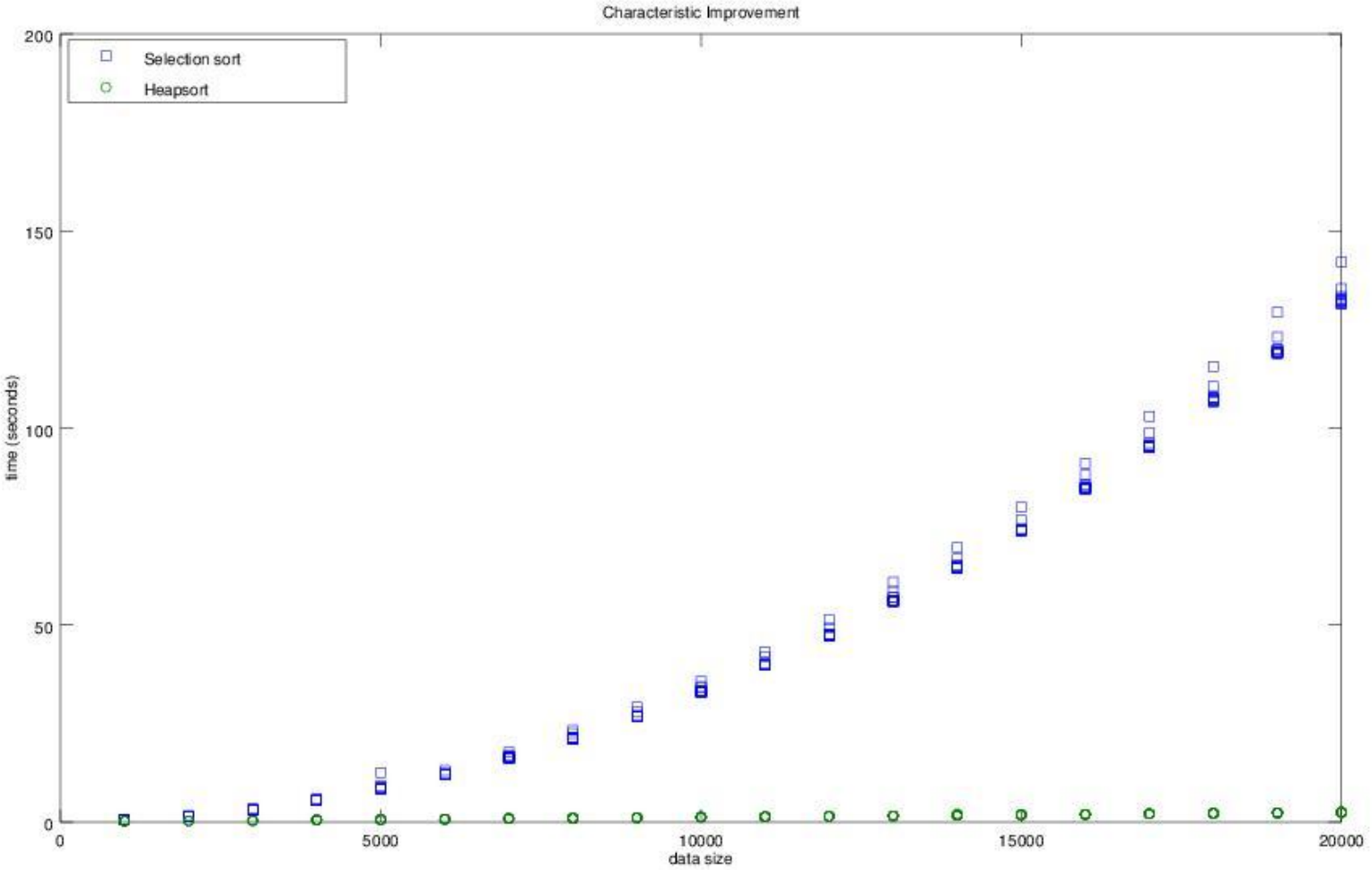


Heap Sort vs. Selection Sort

```
proc HeapSort( arr[1..n] )
    ConstructHeap(arr[1..n])
    i := n
    while i > 1 do
        arr[i] ⇄ arr[1]
        FixHeap(arr[1..i-1], 1)
        i := i - 1
    endwhile
endproc
```

```
proc SelectionSort( arr[1..n] )

    i := n
    while i > 1 do
        imax := FindMax(arr[1..i])
        arr[i] ⇄ arr[imax]
        i := i - 1
    endwhile
endproc
```


Characteristic Improvement
Selection sort
Heapsort
time (seconds)
data size
0
50
100
150
200
0
5000
10000
15000
20000

Kompleksitas HeapSort

Misalkan $n < 2^h$ untuk suatu h, i.e. $n \neq 2^h$

$$\begin{aligned}\sum_{i=2..n} O(lg(i)) &= lg(2)+lg(3)+lg(4)+lg(5)+...+lg(n) \\ &\leq lg(2)+lg(2)+lg(4)+lg(4)+...+lg(½n) \\ &= 2lg(2)+4lg(4)+...+½n\ lg(½n) \\ &\leq n\ lg(½n) \\ &= n\ lg(n) -2n\end{aligned}$$

$$\begin{aligned}T_{HeapSort}(n) &= T_{ConstructHeap}(n) + \sum_{i=2..n} T_{FixHeap}(i,1) \\ &= O(n) + \sum_{i=2..n} O(lg(i)-lg(1)) \\ &= O(n) + \sum_{i=2..n} O(lg(i)-0) \\ &= O(n) + O(n\ lg(n) -2n) \\ &= O(n\ lg(n))\end{aligned}$$

Akhir Bagian 2 dari Topik Minggu 03