SORTING (3)

KULIAH ANALISIS ALGORITMA DAN KOMPLEKSITAS

IMPLEMENTASI HEAP PADA PRIORITY QUEUES

- **Priority queue adalah struktur data untuk menjaga himpunan *S* yang terdiri dari elemen-elemen, dengan setiap elemennya berhubungan dengan suatu nilai yang disebut key.**
- **Heap mengimplementasikan priority queues secara efisien.**
- **Terdapat 2 macam priority queues: max-priority queues dan min-priority queues.**
- **Fokus kita pada max-priority queues, yang berbasis pada max-heaps.**
- **Aplikasi max-priority queues: untuk melakukan penjadwalan jobs pada shared computer.**

OPERASI-OPERASI PADA PRIORITY QUEUES

- **INSERT(*S*, *x*): menyisipkan elemen *x* pada set *S*.**
- **MAXIMUM(*S*): mengembalikan elemen pada *S* dengan key terbesar.**
- **EXTRACT-MAX(*S*): menghapus dan mengembalikan elemen pada *S* dengan key terbesar.**
- **INCREASE-KEY(*S*, *x*, *k*): menaikkan nilai pada elemen berindeks *x* dengan nilai baru *k*. Diasumsikan bahwa *k* ≥ *key* pada indeks x.**

MENCARI ELEMEN TERBESAR

- **MAXIMUM(*S*): memberikan elemen pada *S* dengan key terbesar.**
- **Elemen terbesar pada Max_Heap disimpan di akar.**

HEAP-MAXIMUM(A)

1. return A[1]

- **Running time dari HEAP-MAXIMUM adalah $\Theta(1)$.**

MELAKUKAN EKSTRAKSI TERHADAP ELEMEN TERBESAR

- **EXTRACT-MAX(*S*): menghapus dan mengembalikan elemen pada *S* dengan key terbesar.**

HEAP-EXTRACT-MAX(A)

1. if *heap-size*[A] < 1
2. then error "heap underflow"
3. *max* ← A[1]
4. A[1] ← A[*heap-size*[A]]
5. *heap-size*[A] ← *heap-size*[A] - 1
6. MAX-HEAPIFY(A, 1)
7. return *max*

- **Analisis: assignment dengan waktu konstan + waktu untuk MAX-HEAPIFY.**
- **Running time dari HEAP-EXTRACT-MAX adalah *O*(lg*n*).**

MENAIKKAN NILAI KEY

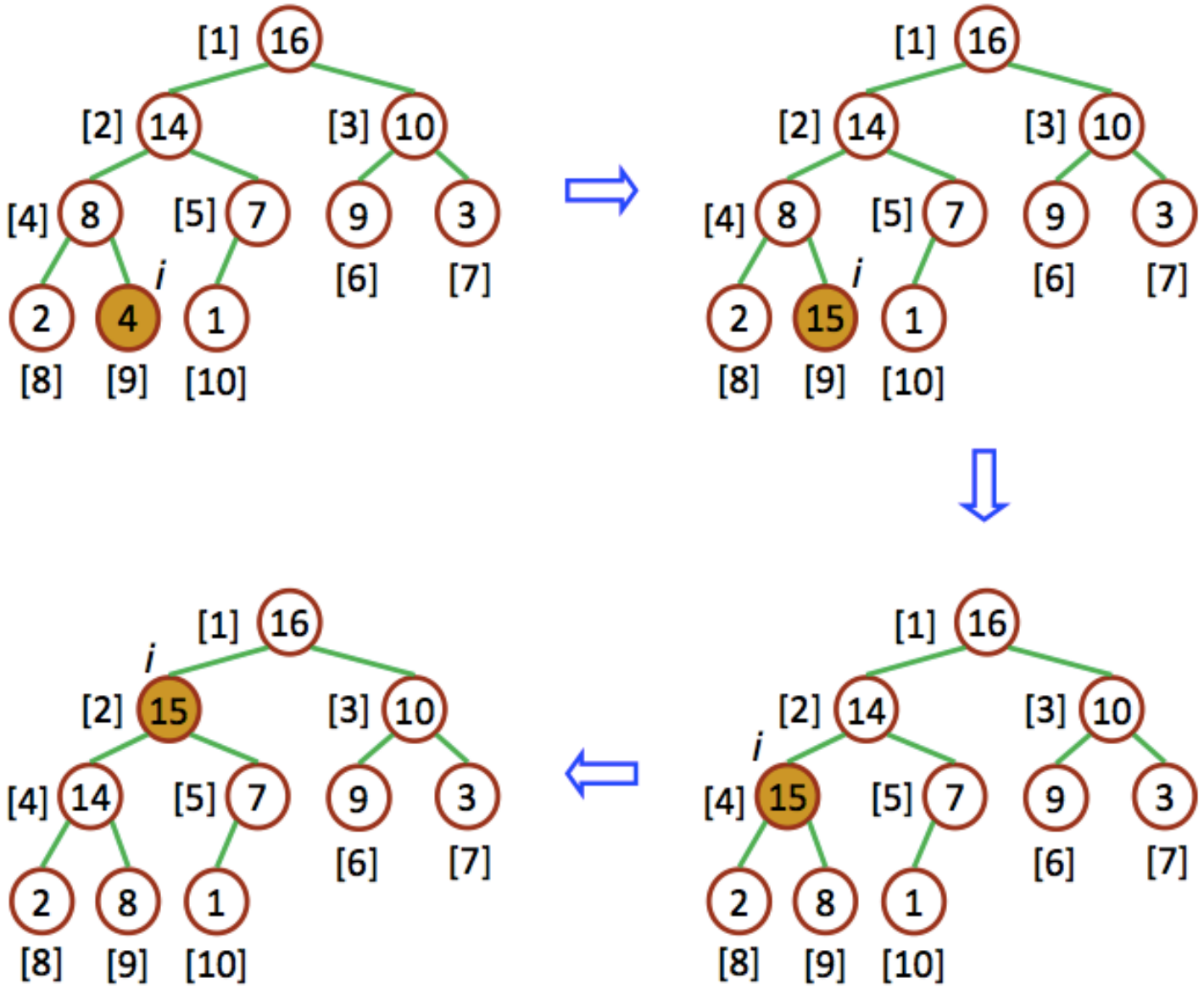
- **INCREASE-KEY(*S*, *x*, *k*): menaikkan nilai pada elemen berindeks *x* dengan nilai baru *k*.**
- **Diasumsikan bahwa *k* ≥ *key* pada indeks x.**

HEAP-INCREASE-KEY (A, *i*, *key*)

1. if *key* < A[*i*]
2. then error "new key is smaller than current key"
3. A[*i*] ← *key*
4. While *i* > 1 and A[PARENT(*i*)] < A[*i*]
5. do exchange A[*i*] ↔ A[PARENT(*i*)]
6. *i* ← PARENT(*i*)

- **Analisis: path dari node yang di-update ke akar memiliki panjang *O*(lg*n*).**
- **Running time : *O*(lg*n*).**

CONTOH



MENYISIPKAN ELEMEN KE HEAP

- **INSERT(*S*, *x*): menyisipkan elemen *x* ke himpunan *S*.**

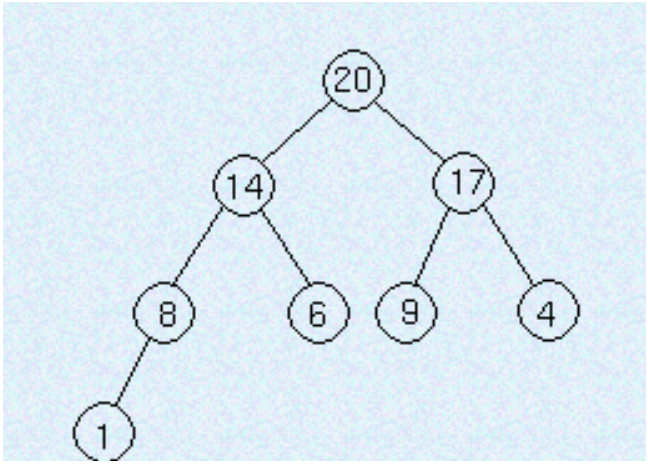
  MAX-HEAP-INSERT(A, *key*)

  1. *heap-size*[A] ← *heap-size*[A]+1
  2. A[*heap-size*[A] ←-∞
  3. HEAP-INCREASE-KEY(A, *heap-size*[A], *key*)

- **Analisis: Waktu konstan assignment + waktu untuk HEAP-INCREASE-KEY.**
- **Running time: *O*(lg*n*).**

LATIHAN

- **Misalkan terdapat heap sbb:**



Sisipkan node dengan key 15 ke heap.