الخوارزميات وبنى المعطيات2

المحاضرة الثانية

د. غيث ابراهيم بلال

الترتيب بالحشر ***insert sort***

بفرض لدينا المصفوفة الجزئية المرتبة $A[\,i-1\,] \ldots\ldots\ldots A[\,1\,]$ من المصفوفة A ونحن **نبحث عن موقع العنصر** $\boldsymbol{A[\,i\,]}$ **ضمن المصفوفة المرتبة** ويتم ذلك بالشكل التالي:

نبدأ من نهاية المصفوفة الجزئية المرتبة $A[\,i-1\,] \ldots\ldots\ldots A[\,1\,]$ بعملية مقارنة العنصر $A[\,i\,]$ مع باقي عناصر المصفوفة السابقة وبتنفيذ عملية إزاحة لقيم العناصر المرتبة حيث يتكرر هذا العمل من 2 حتى n .

نص الخوارزمية:

1. Start.
2. input n , A.
3. i ← 2.
4. while(i ≤ n).
   - 4.1. k ← i – 1.
   - 4.2. key ← A[i].
   - 4.3. While (A[k] > key and k ≥ 1).
     - 4.3.1. A[k + 1] ← A[k].
     - 4.3.2. k ← k – 1.
   - 4.4. A[k + 1] ← key.
   - 4.5. i ← i + 1.
5. End.

- طبق خوارزمية الحشر insert sort لترتيب مصفوفة الاعداد A التالية:

7	-1	10	-2	0	3
1	2	3	4	5	6

i	N	i ≤ n	K	key	A[k] > key
2	6	√	1	-1	7 > -1 √

A[2] = A [1]
A[1] = -1

-1	7	10	-2	0	3

حصلنا على مصفوفة جزئية مرتبة

ثم يصبح

i	N	i ≤ n	K	key	A[k] > key
3	6	√	2	10	×

هنا لا نغير القيم لأن الشرط غير محقق.

نتابع (نبحث عن موقع -2).

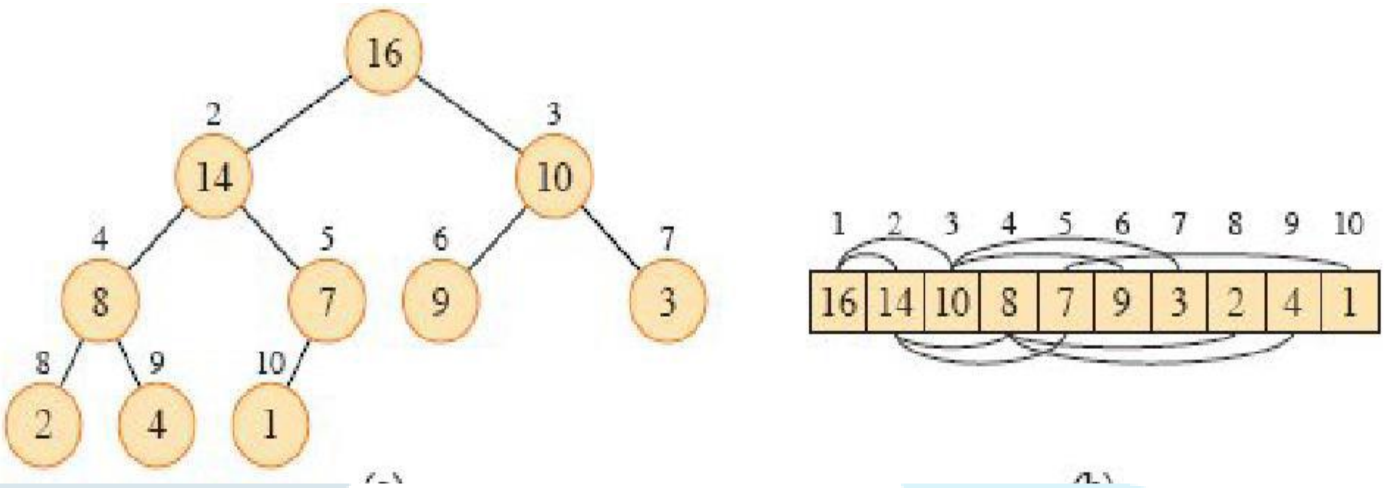
i	N	i ≤ n	K	Key	A[k] > key
4	6	√	3	10	√

نتابع بنفس الأسلوب لنحصل بالنهاية على مصفوفة مرتبة من الاعداد.

-2	**-1**	*0*	3	5	10

إن كلفة هذه الخوارزمية هو $O(n^2)$.

Heaps



Heaps

- An array A[1 : n] that represents a heap is an object with an attribut A.heap-size, which represents how many elements in the heap are stored within array A
- A[1 : A.heap-size], where $0 \le$ A.heap-size $\le n$, are valid elements of the heap
- The root of the tree is A[1]
- given the index i of a node, there's a simple way to compute the indices of its parent, left child, and right child with the procedures PARENT, LEFT, and RIGHT

PARENT(i)
return $\lfloor i/2 \rfloor$

LEFT(i)
return $2i$

RIGHT(i)
return $2i + 1$

max-heap property

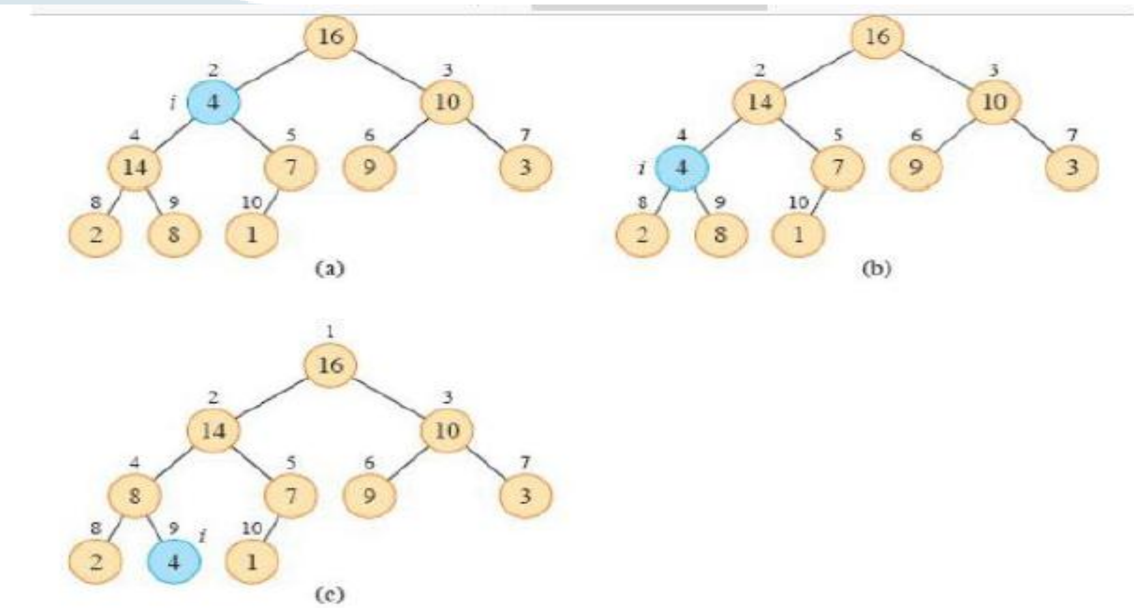
- the max-heap property is that for every node i other than the root, A[PARENT(i)] ≥ A[i]
- largest element in a max-heap is stored at the root
- Viewing a heap as a tree, we define the height of a node in a heap to be the number of edges on the longest simple downward path from the node to a leaf,
- we define the height of the heap to be the height of its root.
- The height of heap containing n elements is Θ(lg n)

Maintaining the max heap property

- The procedure MAX-HEAPIFY maintains the max heap property
- Its inputs are an array A with the heap-size attribute and an index i into the array.
- When it is called, MAX-HEAPIFY assumes that the subtrees rooted at LEFT(i) and RIGHT(i) are max-heaps, but that A[i] might be smaller than its children, thus violating the max- heap property.
- MAX-HEAPIFY lets the value at A[i] “float down” in the max-heap so that the subtree rooted at index i obeys the max-heap property

```
MAX-HEAPIFY(A, i)
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```

we can characterize the running time of MAX-HEAPIFY on a node of height h as O(h).



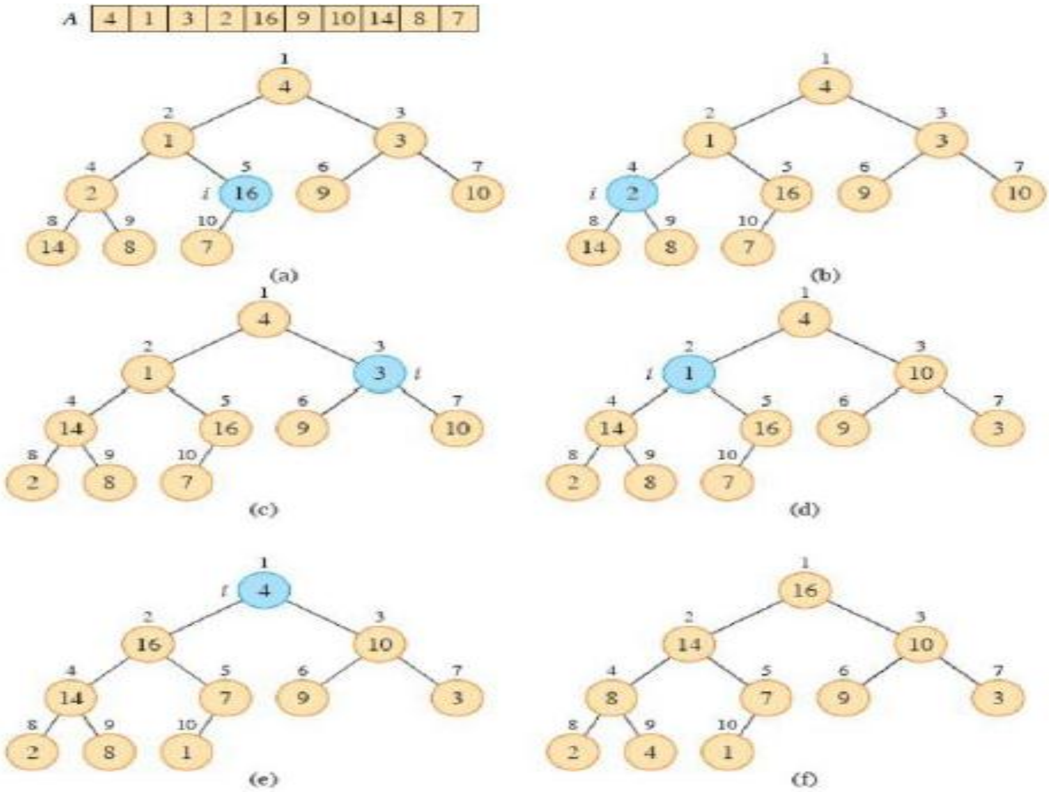
Building a heap

- The procedure BUILD-MAX-HEAP converts an array A[1 : n] into a max-heap

```
BUILD-MAX-HEAP(A, n)
  A.heap-size = n
  for i = ⌊n/2⌋ downto 1
      MAX-HEAPIFY(A, i)
```

We can compute a simple upper bound on the running time of BUILD-MAX-HEAP as follows. Each call to MAX-HEAPIFY costs $O(\lg n)$ time, and BUILD-MAX-HEAP makes $O(n)$ such calls. Thus, the running time is $O(n \lg n)$. This upper bound, though correct, is not as tight as it can be.

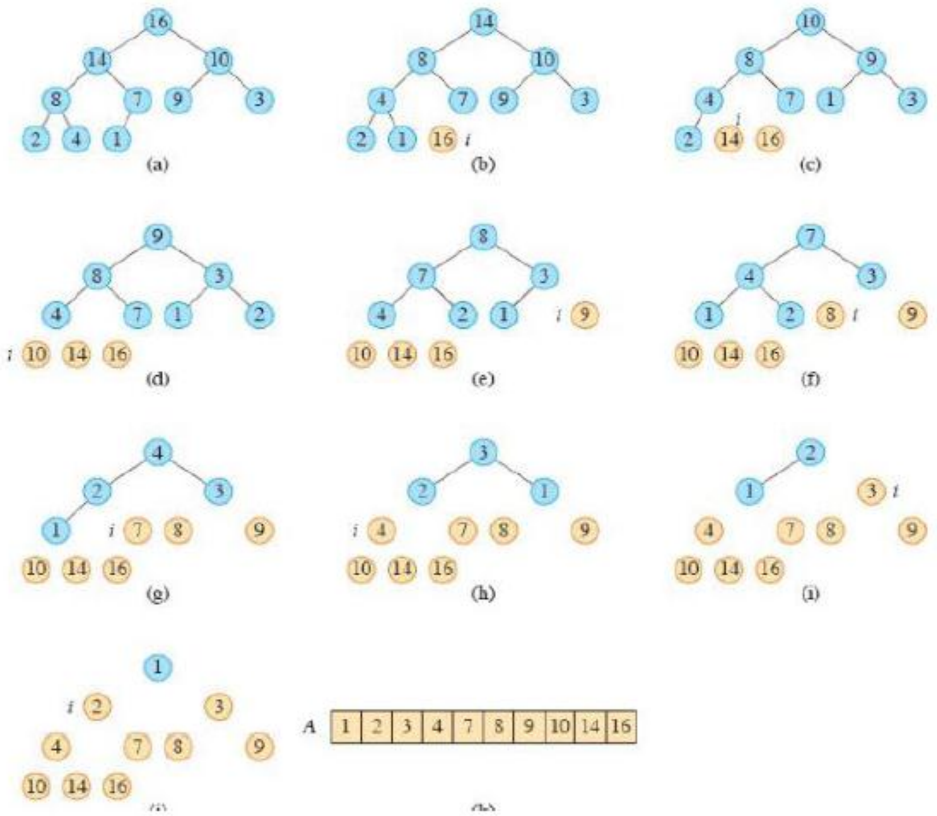
BUILD-MIN-HEAP produces a min-heap from an unordered linear array in linear time.



heapsort

```
HEAPSORT(A, n)
  BUILD-MAX-HEAP(A, n)
  for i = n downto 2
      exchange A[1] with A[i]
      A.heap-size = A.heap-size – 1
      MAX-HEAPIFY(A, 1)
```

The HEAPSORT procedure takes $O(n \lg n)$ time, since the call to BUILD-MAX-HEAP takes $O(n)$ time and each of the $n - 1$ calls to MAX-HEAPIFY takes $O(\lg n)$ time.


(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)
(j)
(k)
A
1 2 3 4 7 8 9 10 14 16