Vinh T. Nguyen. Email: vinh.nguyen@ttu.edu

HOMEWORK 4 – CS5381 Analysis of Algorithm

1. QUESTION 1

Why do we want the loop index i in line 2 of BUILD-MAX-HEAP as shown below to decrease from ⌊length[A]/2⌋ to 2 rather than increase from 1 to ⌊length[A]/2⌋?

```
BUILD-MAX-HEAP(A)
1  heap-size[A] ← length[A]
2  for i ← ⌊length[A]/2⌋ downto 1
3       do MAX-HEAPIFY(A, i)
```

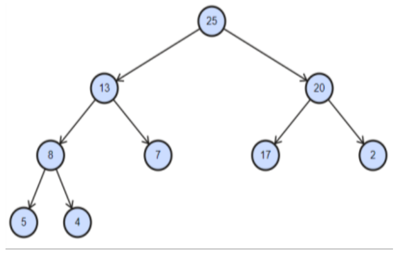
Answer:

Before the first loop iteration, the precondition before calling Max-Heapify must be met in the initialization step (maintain max heap), if we run from step 1, this step is not satisfied. However, running from *⌊length[A]/2⌋,* next nodes (i +1, i+2, ,n) are all leaves and therefore are the root of a trival max-heap. Each iteration also makes sure that both left and right side of subtrees are max heap.

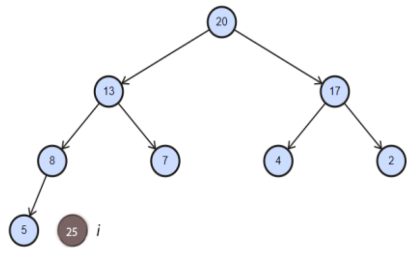
2. QUESTION 2

Using the Figures below as a model, illustrate the operation of HEAPSORT on the array A = ⟨5, 13, 2, 25, 7, 17, 20, 8, 4⟩.

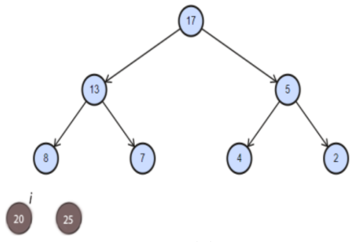
Answer:



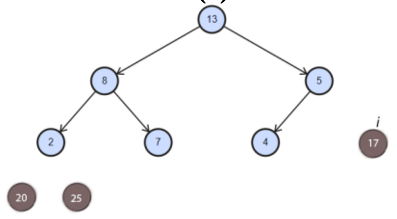
(a)



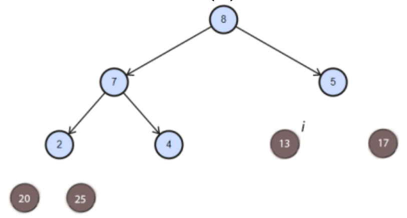
(b)



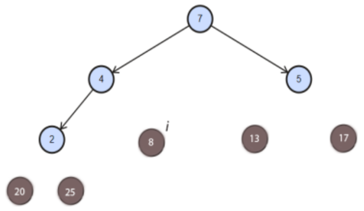
(c)



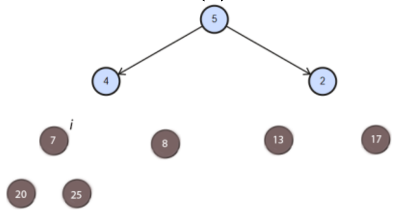
(d)



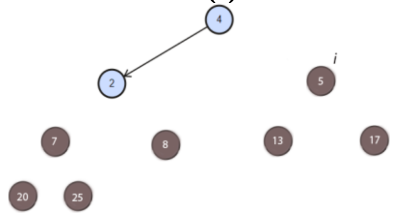
(e)



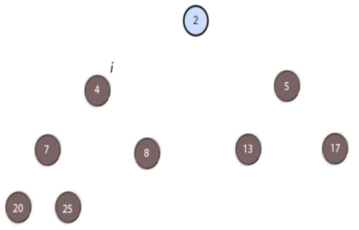
(f)



(g)



(h)



(i)

A = [2, 4, 5, 7, 8, 13, 17, 20, 25]
(j)

3. QUESTION 3

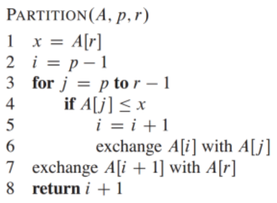
What is the running time of heapsort on an array A of length n that is already sorted in increasing order? What about decreasing order?

Answer:

The running time of heapsort on an array A of length n that is already sorted in increasing order is θ (*n*log*n*) because the order of numbers in the heap is different from the sorted array. Therefore, the sorted array will still be transformed into a heap and do the sorting.

The running time for sorted array in decreasing order is still θ (*n*log*n*) although time for building the heap is linear, when extracting the max heap, max is swapped with the last leaf and the HEAPIFY is called again. This call will cover the whole tree.

4. QUESTION 5



```
PARTITION(A, p, r)
1  x = A[r]
2  i = p - 1
3  for j = p to r - 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

What value of q does PARTITION return when all elements in the array A[p..r] have the same value?

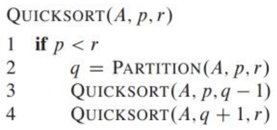
Answer:

This case is similar to the previous problem where line 4 is satisfied for all iterations. *i* increases to r-1 in the last iteration and then increased by 1 with the return (*i*+1). So, the value of *q = (r-1+1) = r*

5. QUESTION 6

What is the running time of QUICKSORT when all elements of array A have the same value?

Answer:



```
QUICKSORT(A, p, r)
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q - 1)
4      QUICKSORT(A, q + 1, r)
```

The worst-case running time occurs when q=r, meaning that the partition produces one subproblem with size n-1 elements and the other subproblem size 0. The partitioning cost θ(n) time, the recursive call on the array size T(0) is θ(1). Thus, the recurrence for the running time is:

T(n) = T(n-1) + T(0) + θ(n) = T(n-1) + θ(n)

Using substitution method we have

T(n) = T(n-1) + θ(n)
= T(n-2) + θ(n-1) + θ(n)
= T(n-3) + θ(n-2) + θ(n-1) + θ(n)
= T(n-k) + θ(n-k+1) +….+ θ(n)

Set n-k=0 => n=k we have

T(n) = T(0) + θ(1) + θ(2)+ θ(3)+…+ θ(n)
= T(0) + θ(1+2+3+…+n)
= T(0) + θ(n(n+1)/2)= θ(1) + + θ(n(n+1)/2)
= θ(n^2)

Therefore, the running time of QUICKSORT when all elements of array have the same value is **θ(n^2)**

6. QUESTION 7

Illustrate the operation of COUNTING-SORT on the array A = ⟨6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2⟩.

A = [6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2]

index	1	2	3	4	5	6	7	8	9	10	11
A	**6**	**0**	**2**	**0**	**1**	**3**	**4**	**6**	**1**	**3**	**2**

C = [2, 2, 2, 2, 1, 0, 2]

index	0	1	2	3	4	5	6
C	**2**	**4**	**6**	**8**	**9**	**9**	**11**

index	1	2	3	4	5	6	7	8	9	10	11
B											

1 iteration: j=11
A[11] =2, C[2]=6, B[6]=A[11]=2. C[2]=>5

index	1	2	3	4	5	6	7	8	9	10	11
B						**2**					

index	0	1	2	3	4	5	6
C	**2**	**4**	**5**	**8**	**9**	**9**	**11**

2 iteration: j=10
A[10] =3, C[3]=8, B[8]=A[10]=3. And C[3]=>7

index	1	2	3	4	5	6	7	8	9	10	11
B						**2**		**3**			

index	0	1	2	3	4	5	6
C	**2**	**4**	**5**	**7**	**9**	**9**	**11**

3 iteration: j=9
A[9] =1, C[1]=4, B[4]=A[9]=1. And C[1]=>3

index	1	2	3	4	5	6	7	8	9	10	11
B				**1**		**2**		**3**			

index	0	1	2	3	4	5	6
C	**2**	**3**	**5**	**7**	**9**	**9**	**11**

4 iteration: j=8
A[8] =6, C[6]=11, B[11]=A[8]=6. And C[6]=>10

index	1	2	3	4	5	6	7	8	9	10	11
B				**1**		**2**		**3**			6

index	0	1	2	3	4	5	6
C	**2**	**3**	**5**	**7**	**9**	**9**	**10**

5 iteration: j=7
A[7] =4, C[4]=9, B[9]=A[7]=4. And C[4]=>8

index	1	2	3	4	5	6	7	8	9	10	11
B				**1**		**2**		**3**	**4**		6

index	0	1	2	3	4	5	6
C	**2**	**3**	**5**	**7**	**8**	**9**	**10**

6 iteration: j=6
A[6] =3, C[3]=7, B[7]=A[6]=3. And C[3]=>6

index	1	2	3	4	5	6	7	8	9	10	11
B				**1**		**2**	**3**	**3**	**4**		6

index	0	1	2	3	4	5	6
C	**2**	**3**	**5**	**6**	**8**	**9**	**10**

7 iteration: j=5

A[5] =1, C[1]=3, B[3]=A[5]=1. And C[1]=>2

index	1	2	3	4	5	6	7	8	9	10	11
B			**1**	**1**		**2**	**3**	**3**	**4**		6

index	0	1	2	3	4	5	6
C	**2**	**2**	**5**	**6**	**8**	**9**	**10**

8 iteration: j=4

A[4] =0, C[0]=2, B[2]=A[4]=0. And C[0]=>1

index	1	2	3	4	5	6	7	8	9	10	11
B		**0**	**1**	**1**		**2**	**3**	**3**	**4**		6

index	0	1	2	3	4	5	6
C	**1**	**2**	**5**	**6**	**8**	**9**	**10**

9 iteration: j=3

A[3] =2, C[2]=5, B[5]=A[3]=2. And C[2]=>4

index	1	2	3	4	5	6	7	8	9	10	11
B		**0**	**1**	**1**	**2**	**2**	**3**	**3**	**4**		6

index	0	1	2	3	4	5	6
C	**1**	**2**	**4**	**6**	**8**	**9**	**10**

10 iteration: j=2

A[2] =0, C[0]=1, B[1]=A[2]=0. And C[0]=>0

index	1	2	3	4	5	6	7	8	9	10	11
B	**0**	**0**	**1**	**1**	**2**	**2**	**3**	**3**	**4**		6

index	0	1	2	3	4	5	6
C	**0**	**2**	**4**	**6**	**8**	**9**	**10**

11 iteration: j=1

A[1] =6, C[6]=10, B[10]=A[1]=6. And C[6]=>9

index	1	2	3	4	5	6	7	8	9	10	11
B	**0**	**0**	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**6**	6

index	0	1	2	3	4	5	6
C	**0**	**2**	**4**	**6**	**8**	**9**	**9**

7. QUESTION 11

Using the Figure below as a model, illustrate the operation of RADIX-SORT on the following list of English words: COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX

COW	SEA	TAB	BAR
DOG	TEA	BAR	BIG
SEA	MOB	EAR	BOX
RUG	TAB	TAR	COW
ROW	DOG	SEA	DIG
MOB	RUG	TEA	DOG
BOX	DIG	DIG	EAR

TAB	BIG	BIG	FOX
BAR	BAR	MOB	MOB
EAR	EAR	DOG	NOW
TAR	TAR	COW	ROW
DIG	COW	ROW	RUG
BIG	ROW	NOW	SEA
TEA	NOW	BOX	TAB
NOW	BOX	FOX	TAR
FOX	FOX	RUG	TEA

8. QUESTION 12

Show how to sort n integers in the range 0 to $n^3 - 1$ in O(n) time.

Answer:

First, given a number in the range [0, N-1], the number of digits to represent this number in the base-b system is

d = ceiling($\log_b N$). Replace N by n^3 we have d = ceiling($\log_b n^3$) = ceiling($3\log_b n$). For simplifying the result to get rid of log, let set base b =n. Then we have d = ceiling (3) = 3.

Second, given *n* d-digits numbers, each digit can take up to k possible values. RADIX-SORT can sort correctly in θ (*d*(*n*+*k*)) time if n is much larger than k. This can be easily proved by using counting sort for the least influence column first (right to left). Each column takes θ(n+k) time, we have *d* columns (or digits), so the total running time is θ(d*(n+k))

Because we set based b = n, so k ranges from 0 to n-1 => *n* possible values.

So the total running time is T(n) = θ(d*(n+k)) = θ(3*(n+n)) = O(n).