ASHUTOSH D. MODI (modia)

96

Introduction to Theory of Algorithms, Midterm I. This is a closed book, closed notes exam and no calculators are allowed. Sep. 28th, 2016

Questions 1 (16 points): Fill in the following blanks to finish the MERGE function (2 points for each blank).

Q2a -2

Q4 -2

MERGE *(A, p, q, r)*

n_1 = q - p + 1

n_2 = r - q

Create arrays $L[1..(n_1+1)]$ and $R[1..(n_2+1)]$

for $i = 1$ **to** n_1

$L[i]$ = A[p+i-1]

for $j = 1$ **to** n_2

$R[j]$ = A[q+j]

$L[n_1+1]$ = ∞ ;

$R[n_2+1]$ = ∞

$i = 1$; $j = 1$

for $k = p$ **to** r

if $L[i] \le R[j]$

then A[k] = L[i] ; ++i;

else A[k] = R[j]; ++j;

Question 2 (24 points): Solve the recurrence relations below.

-2 a. (6 points): $T(n) = 2T(n-2) + \sqrt{n}$

$\sqrt{n}$ $\quad 2^0\sqrt{n-2^0}$

$\sqrt{n-2}$ $\quad \sqrt{n-2}$ $\quad 2^1\sqrt{n-2^1}$

$\sqrt{n-4} \ldots$ $\quad \sqrt{n-4}$ $\quad 2^2\sqrt{n-2^2}$

Stop condition $n - 2^k = 0$

$n = 2^k \Rightarrow k = \lg_2 n$

last levels cost $= 2^k\sqrt{n-2^k}$

So total cost for T(n)

$= \sum_{i=0}^{k} 2^i\sqrt{n-2^i}$

(Assume n is power of 2)

$= \sum_{i=0}^{\lg_2 n} 2^i\sqrt{n-2^i} \le \sum_{i=0}^{\lg n} 2^i \cdot \sum_{i=0}^{\lg n} \sqrt{n-2^i}$

(Since we need to find upper bound)

$\Rightarrow = \left(\frac{2^{\lg n+1}-1}{1}\right) \cdot \sum_{i=0}^{\lg n}\sqrt{n-2^i}$

Can be approximated to.

$= 2^{\lg n+1} \cdot \sum_{i=0}^{\lg n}\sqrt{n} = 2 \cdot 2^{\lg n} \cdot \sqrt{n} \cdot \frac{\lg n(\lg n+1)}{2}$

$= 2\sqrt{n} \cdot n^{\lg 2} \cdot \left(\frac{(\lg n)^2+\lg n}{2}\right) = \sqrt{n} \cdot n^1\left((\lg n)^2+\lg n\right)$

$\Rightarrow T(n) \in O\left(n^{3/2}(\lg n)^2\right)$ X

b. (6 points): $T(n) = 5T(n/4) + \lg n$

$f(n) = \lg n$ $a = 5$ $b = 4$ & $f(n) = \lg n$

$\because$ $a \geq 1$ & $b > 1$ $\Rightarrow$ $\log_b^a = \log_4^5$

By M.T. case I.

$f(n)$ i.e. $\lg n \in O(n^{\log_4^5 - \varepsilon})$

for small ε value this holds true & $\varepsilon > 0$

$\Rightarrow \Theta(n^{\log_b^a})$

$T(n) \in \Rightarrow \Theta(n^{\log_4^5})$

c. (6 points): $T(n) = 4T(n/4) + n\lg^2 n$

$f(n) = n \lg^2 n$; $a = 4$; $b = 4$

$\because$ $a \geq 1$; $b > 1$ & $\log_b^a = 1$

Using M.T. Case II which is a generic form

$f(n) \in \Theta(n^{\log_b^a} \cdot \log^k n) \Rightarrow \Theta(n^{\log_b^a} \log^{k+1} n)$

in our case

$f(n) \in \Theta(n^1 \log^2 n)$ where $k = 2$

$\Rightarrow$ $T(n) \in \Theta(n \log^3 n)$

d. (6 points): $T(n) = 2T(n/4) + \sqrt{n}\lg n$

$a = 2$, $b = 4$, $f(n) = \sqrt{n}\ \lg n$ $\therefore a \geq 1, b > 1$

$\Rightarrow \log_b a = \log_4 2 = 1/2$

Using. M.T. Case $\underline{II}$ generic form

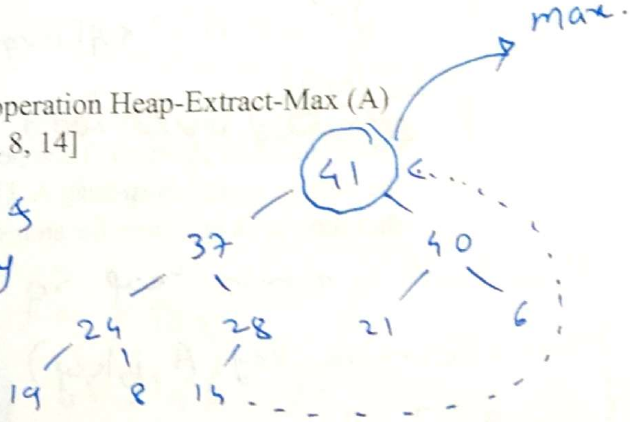
$f(n) \in \Theta(n^{\log_b a} \log^k n) \Rightarrow T(n) \in \Theta(n^{\log_b a} \log^{k+1} n)$

$\because f(n) \in \Theta(n^{1/2} \log n)$ where $k = 1$

$\Rightarrow T(n) \in \Theta(n^{\log_4 2} \log^2 n)$

$\Rightarrow T(n) \in \Theta(\sqrt{n}\ \lg \lg n)$

Question 3 (15 points) Show the steps and result of the operation Heap-Extract-Max (A) on the following array: A = [41, 37, 40, 24, 28, 21, 6, 19, 8, 14]



This is a Max Heap & nearly complete binary tree.

$\therefore$ No need to call build max Heap function.

Heap Extract Max (A)
{ if size < 1 then "error"
max = A[1]
A[1] = A[size]
size = size − 1
Max Heapify (A, 1)
return max;
}

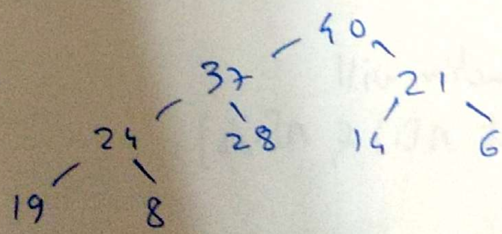
Step 1: [14, 37, 40, 24, 28, 21, 6, 19, 8] ← size has been reduced

MaxHeapify.

Step 2: [40, 37, 14, 24, 28, 21, 6, 19, 8]

Step 3: [40, 37, 21, 24, 28, 14, 6, 19, 8]

So heap looks like,



Scanned by CamScanner

-2 **Question 4** (15 points): Can MAX-Heapify be implemented in a non-recursive way? If your answer is yes, please write the implementation code. If your answer is no, please explain why. Your answer to this question should point out Yes or No.

Max-Heapify can be implemented in non-recursive way if it is executed in while ~~for~~ or for loop with checking conditions.

```
Max-Heapify(A)
{   while true
  /*for (i = ⌊A.length/2⌋ to 1)*/
    { l=2i , r=2i+1
      if(l ≤ A.size & A[l] > A[i])
          then large = l ;
      else  large = i ;
      if (r ≤ A.size & A[r] > A[large]
            large = r
      if (large != i)
          {swap A[i] & A[large]   {else break}
      i = large
    }   overwritted by for loop
```

This pseudo code builds max heap by calling operations in a for loop.

Questions 5 (15 points): The operation HEAP-DECREASE-KEY (A, i, key) decreases the item in node *i* from heap A. Give an implementation of HEAP-DECREASE-KEY that runs in O(lgn) time for an n-element min-heap.

Heap built is a min heap so to decrease key can be implemented as below.

```
Heap-Decrease-key(A,i,key)
{
    A[i] = key
    while (i>1 && A[i/2] > A[i]
     {
        Swap (A[i], A[i/2])
           i = i/2 ;
     }
}
```

In this pseudo code while loop is executed for i>1 & i is decreasing by a factor of 2 i.e. i = i/2

⇒ time complexity is O(lgn)

where swap function will exchange elements at A[i] & A[i/2]

Scanned by CamScanner

Questions 6 (15 points) Given unsorted arrays A and B, we use the Merge function to merge both arrays. Is the merged result a sorted array? Please first answer Yes or No and then explain your answer.

If arrays A & B have only 1 elements in each then Merge function will give sorted merged array.

Very thoughtful !

Otherwise Merge function will not give sorted arrays.

Consider following Merge Sort Subroutine :

```
MSort ( A,p,q,r)
{ if (p<r)
    q = ⌊(p+r)/2⌋
    MSort (A,p,q)
    MSort ( A,q+1,r)
    Merge (A,p,q,r)
}
```

From this function it is clear that Merge function needs sorted arrays to merge.

Moreover consider the below snippet of main code wherein comparison is being done on left and right arrays. (in our case it is A & B)

```
for k = p to r
{   if L[i] ≤ R[j]
        A[k] = L[i]
        ++i;
    else A[k] = R[j]
        ++j;
}
```

From this for loop it is clear that elements are compared sequentially & not checked with previous elements. Making it ~~to~~ fail to create sorted merged Arrays.