Design and Analysis of Computer Algorithms

Homework #3: This homework is not graded, it is used as a preparation for midterm

Problem #1. What are the minimum and maximum numbers of elements in a heap of height h?

Solution:

- Heap is a complete binary tree
- Number of nodes at level $i < h$ is 2^i
- From level 0 to level $(h-1)$, there are $\sum_{i=0}^{h-1} 2^i = 2^h - 1$
- At level h, number of nodes is from 1 to 2^h
- Minimum and maximum numbers of elements of the heap are 2^h and $(2^{h+1} - 1)$

Problem #2. Why do we want the loop index i in line 2 of BUILD-MAX-HEAP to decrease from $\lfloor A.length/2 \rfloor$ to 1 rather than increase from 1 to $\lfloor A.length/2 \rfloor$?

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

Solution:

If we had started at 1, we wouldn't be able to guarantee that the max-heap property is maintained. For example, if the array A is given by [2,1,1,3] then MAX-HEAPIFY won't exchange 2 with either of it's children, both 1's. However, when MAX-HEAPIFY is called on the left child, 1, it will swap 1 with 3. This violates the max-heap property because now 2 is the parent of 3.

Problem #3. Solve the following recurrences:

a. $T(n) = 2T(n/2) + n^4$
b. $T(n) = T(7n/10) + n$
c. $T(n) = 16T(n/4) + n^2$
d. $T(n) = 7T(n/3) + n^2$
e. $T(n) = 7T(n/2) + n^2$
f. $T(n) = 2T(n/4) + \sqrt{n}$

Hint: Apply Master theorem, we obtain the following results

a. $T(n) \in \Theta(n^4)$.
b. $T(n) \in \Theta(n)$.
c. $T(n) \in \Theta(n^2 \log n)$.
d. $T(n) \in \Theta(n^2)$.

e. $T(n) \in \Theta(n^{\log(7)})$.

f. $T(n) \in \Theta(n^{\frac{1}{2}} \log n)$.

Problem #4. Given the following recursive ***factorial (int n)*** function as following. Derive the recurrence relation and determine the time complexity of that function.

```
int factorial(int n)
{
    if (n == 0)
    {
        return 1;
    }
    return n * factorial(n-1);
}
```

Hint:

When $n > 1$, the function performs a fixed number of operations, and makes a recursive call to *factorial(n-1)*. This recursive call will perform $T(n-1)$.

Recurrence relation for above code is:

$$\begin{cases} T(1) = 1 \\ T(n) = 1 + T(n-1),\ for\ n > 1 \end{cases}$$

Use substitution method, we have $T(n) \in \Theta(n)$.

Problem #5. The worst-case partitioning of quicksort has the recurrence $T(n) = T(n-1) + \theta(n)$. Use the substitution method to prove that the recurrence has the solution $T(n) = \theta(n^2)$.

Problem #6. Illustrate the operation of COUNTING-SORT on the array A = {6,0,2,0,1,3,4,6,1,3,2}.

Hint:

Using Figure 8.2 of textbook (slide 17 of lecture note) as a model

Problem #7. Illustrate the operation of RADIX-SORT on the following list of words: COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX.

Hint:

Using Figure 8.3 of textbook (slide 33 of lecture note) as a model

Problem #8. Illustrate the process of inserting the keys 10, 22, 31, 4, 15, 28, 17, 88, 59 into a hash table of length m=11 using linear probing, the primary hash function is $h_1(k) = k$.

Hint:

Hash function *h(k,i) = (h'(k) + i)mod m = (k+i) mod m*

Check slide #45 of lecture note for a similar example. The final result is:

T[10] = 10

T[0] = 22

T[9] = 31

T[4] = 4

T[5] = 15

T[6]= 28

T[7] = 17

T[1] = 88

T[8] = 59

Problem #9. We are given a **sorted** array **A** of **n** elements and a value *v***.** Our target is to output index *i* such that *v* = **A**[*i*] or the special value **NULL** if *v* does not appear in **A**. We use the following algorithm to solve that problem: We check the midpoint of **A** against *v* and eliminate half of **A** from further consideration. We repeat this procedure until finding the output. Write the pseudo code and prove that the worst-case time complexity is $\Theta(lg\ n)$.

Hint: The algorithm above is binary search (BinSearch).

```
BinSearch(a,b,v)
if      a > b
    return NIL
end if
m = ⌊(a+b)/2⌋
if      A[m]= v
    return m
end if
if      A[m]< v
    return BinSearch(a,m,v)
end if
return BinSearch(m+1,b,v)
```

From the pseudo code, we have recurrence relation of BinSearch is:

$$T(n) = T(n/2) + O(1)$$

Apply case 2 of master method, we have $T(n) = \Theta(lg\ n)$.