DESIGN AND ANALYSIS OF ALGORITHMS – CSCI 612

NAME: AADITHYA GOPALAKRISHNA BHARADWAJ

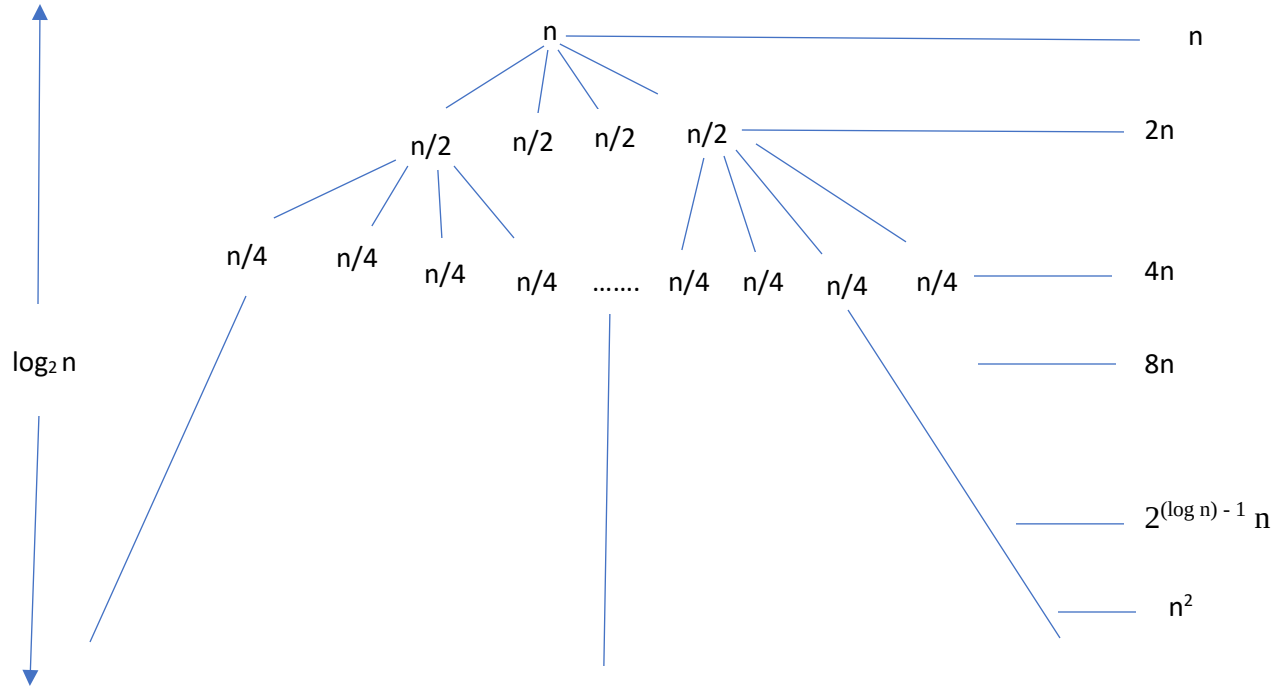
ZID: Z1862641

ASSIGNEMENT NUMBER, SEMESTER: Assignment 3, SPRING 2019

==========

1.) a) Draw the recursion tree for T(n) = 4T(n/2) + n and give a tight asymptotic bound on its solution.
b) Verify your bound by induction. c) Use the master's method to solve above recurrence.

Answer.) a.) The recursion for T(n)=4T(n/2)+n is as follows:



b.) Since T(n) = the sum of recursion tree

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

$= n^2 + n\ (2^{\lg n} - 1)$

$= n^2 + n^2 - n$

$= \Theta(n^2)$

c.) Using master's method to verify the answer:

a = 4, b = 2, f(n) = n, $n^{\lg_b a} = n^{\lg_2 4} = n^2$

So $f(n) = O(n^{\lg_b a})$

Hence applying master method, we have $T(n) = \Theta(n^2)$

2.) Use the master method to solve $T(n) = aT(n/2) + cn^3$ for three cases a = 7, 8, 9 and a constant c > 0.

Answer.) The Master Theorem applies to recurrences of the following form: $T(n) = aT(n/b) + f(n)$ where $a \geq 1$ and $b > 1$ are constants and f(n) is an asymptotically positive function.

There are 3 cases:

1. If $f(n) = O(n^{\log_b a - c})$ for some constant c > 0, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a} \log^k n)$ with[1] $k \geq 0$, then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$.
3. If $f(n) = \Omega(n^{\log_b a + c})$ with c > 0, and f(n) satisfies the regularity condition, then $T(n) = \Theta(f(n))$. Regularity condition: $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n.

Now considering our question, we have three cases:

➔ a=7, then the equation becomes as follows:
- $T(n) = aT(n/2)+cn^3$
- ⇨ $T(n) = 7T(n/2)+cn^3$
- ⇨ Here a = 7, b = 2
- ⇨ $n^{\log_2 7}$
- ⇨ Now $n^{2.8} < f(n) = O(n^3)$,
- ⇨ Hence by method 3 of master's theorem, $T(n) = \Theta(n^{2.8}) \sim \Theta(n^3)$

➔ a=8, then the equation becomes as follows:
- $T(n) = aT(n/2)+cn^3$
- ⇨ $T(n) = 8T(n/2)+cn^3$
- ⇨ Here a = 8, b = 2, since c > 0
- ⇨ $n^{\log_2 8}$
- ⇨ $T(n) = \Theta(n^{\log_2 8}) = f(n) = O(n^3)$
- ⇨ Hence by method 2 of master's theorem, $\Theta(f(n)\log n) \Rightarrow T(n) = \Theta(n^3 \log n)$

➔ a=9, then the equation becomes as follows:
- $T(n) = aT(n/2)+cn^3$
- ⇨ $T(n) = 8T(n/2)+cn^3$
- ⇨ Here a = 9, b = 2, since c > 0, applying
- ⇨ $(n^{\log_2 9})$
- ⇨ Now $n^{3.16} > f(n) = O(n^3)$
- ⇨ Hence by method 1 of master's theorem, $T(n) = O(n^{3.16})$

3.) You are given an unsorted array of n integers.

1) What is the time cost to find the maximum or minimum?

2) Assume the time cost to find each of the maximum and minimum is T(n). A straightforward way to find both of the maximum and minimum costs 2T(n). Please give a better approach using divide-and conquer. Describe the recurrence and solve it. You can assume n is power of 2.

Answer.) 1.) The time complexity to find the maximum and minimum in an unsorted array is ***O*(n)**.

2.) Divide the array into two parts and compare the minimums and maximums of the respective parts to get the minimum and maximum of the whole array.

Algorithm:

```
PairMaxMin(array, array_size)
    if array_size = 1
        return element as both maximum and minimum
    else if array_size = 2
        one comparison to determine the maximum and minimum
        return that pair
    else    /* array size > 2 */
        recurrence for maximum and minimum of left half
        recurrence for maximum and minimum of right half
        one comparison determines true maximum of the two candidates
        one comparison determines true minimum of the two candidates
        return the pair of maximum and minimum
```

Time complexity is ***O*(n)**

Total number of comparisons: Let number of comparisons be T(n). T(n) can be written as follows:

$T(n) = T(\text{floor } (n/2)) + T(\text{ceil } (n/2)) + 2$
$T(2) = 1$
$T(1) = 0$

If n is power of 2, then we can write T(n) as:

$T(n) = 2T(n/2) + 2$
$\Rightarrow T(n) = (2T(n/4)+2) + 2 + \ldots.$

After solving above recursion, we get

$T(n) = (3n/2) - 2$

Thus, the approach does (3n/2) – 2 if n is power of 2 and it does more than (3n/2) -2 comparisons if n is not power of 2.

4.) Banks often record transactions on an account in order of the times of the transactions, but many people like to receive their bank statement with checks listed in order by check number. People usually write checks in order by check number, and merchants usually cash them with reasonable dispatch. The problem of converting time-of-transaction ordering to check-number ordering is therefore the problem of sorting almost-sorted input. Argue that insertion sort would tend to beat quick sort on this problem.

Answer.) INSERTION-SORT's running time on perfectly-sorted input runs in $\Theta(n)$ time. So, it takes almost $\Theta(n)$ running time to sort an almost-sorted input with INSERTION-SORT.

However, QUICKSORT requires almost $\Theta(n^2)$ running time, recalling that it takes $\Theta(n^2)$ time to sort perfectly-sorted input. As the data being sorted becomes closer to sorted, insertion sort gets closer to $Q(n)$ and quicksort gets closer to $Q(n^2)$. This is because when we pick the last element as the pivot, it is usually the biggest one, and it will produce one subproblem with close to $n - 1$ elements and one with 0 elements.

Since the cost of PARTITION procedure of QUICKSORT is $\Theta(n)$, the recurrence running time of QUICKSORT is $T(n) = T(n - 1) + \Theta(n)$. Therefore, there comes a point at which insertion sort performs better than quicksort. Further, while quicksort is still $Q(n \lg n)$ the constant multiple continues to grow as the data is more sorted. And with insertion sort, the constant multiple continues to shrink.

In another problem, we, use the substitution method to prove that the recurrence $T(n) = T(n - 1) + \Theta(n)$ has the solution $T(n) = \Theta(n^2)$. So, we use INSERTION-SORT rather than QUICKSORT in this situation when the input is almost sorted.

==========

5.) Let A[1…n] be an array of n distinct numbers. If i < j and A[i] > A[j], then the pair (i, j) is called an inversion.

a) List the inversions in the array.

b) What array with elements from the set {1, 2, ,.. , n} has the most number of inversions? How many does it have?

Answer.) a) The definition of an inversion is as follows: Let A[1...n] be an array of n distinct numbers. If i < j and A[i] > A[j], then the pair (i, j) is called an inversion of A.

The five inversions of the array < 2, 3, 8, 6, 1 > are as follows:

(2, 1), (3,1), (8, 6), (8, 1) and (6, 1)

b) The n-element array with most inversions is <n, n-1, … 2,1> because it will be the reverse sorted array. The number of inversions are as follows: (n-1) + (n-2) + (n-3) +…+2+1 = n(n-1)/2.