Semester 4TH
(Regular&Back)
Sub & Code-DAA(CS2008)
Branch (s) -ECS

SPRING MID SEMESTER EXAMINATION-2017

DESIGN & ANALYSIS of ALGORITHM
[CS 4001]

Full Marks: 25 **Time: 1.5 Hours**

Answer any six questions including question No.1 which is compulsory.
The figures in the margin indicate full marks.
Candidates are required to give their answers in their own words as far as practicable and all parts of a question should be answered at one place only.

DAA SOLUTION AND EVALUTION SCHEME

Q No:	Contents	Marks

1. Answer the following: [1x5]

a. Briefly state the properties of Algorithm?

Scheme: correct answer: 1 mark Other answer: 0 mark

Answer: an algorithm must satisfy five properties.

- Input: The input must be specified. That is, the inputs that we are going to take through our algorithm must come from a specified set of elements, and the type and amount of inputs must be specified.
- Output: The output must also be specified. The algorithm must specify what the output should be, and how it is related to the inputs.
- Definiteness: The steps in the algorithm must be definite. In other words, the steps need to be clearly defined and detailed. The action of each step must be specified.
- Effectiveness: The steps in the algorithm must be effective, so they must be doable.
- Finiteness: The algorithm must be finite. That is, the algorithm must come to an end after a specific number of steps.

b. **Solve the recurrence relation using Master theorem. $T(n)=16T(n/4)+n^2$.**

Scheme: correct answer: 1 mark Other answer: 0 mark

Answer: $a = 16, b=4, f(n)=n^2$

$T(n)=O(n^2 \log n)$

c. **Find the time complexity of given code snippet:**

```
int fun(int n)
{   int count = 0;
```

```
for (int i = n; i > 0; i /= 2) for (int j = 0; j < i; j++)
count += 1;
Return count;    }
```

Scheme: correct answer: 1 mark Other answer: 0 mark

Answer: O(n)

d. **Arrange the following functions in increasing order of their growth rate.**

***n*!, 2^n ,*n*log*n* ,n^3, $n^{0.5}$**

Scheme: correct answer: 1 mark Other answer: 0 mark

Answer: $n^{0.5}$, nlogn, n^3,2^n, n!

e. **Match the followings**

(A) O (logn)	**(p) Best-case Linear Search**
(B) O (n)	**(q) Merging of two arrays**
(C) O (1)	**(r) Binary search**
(D) O (n^2)	**(s) Worst-case Insertion sort**

Scheme: correct answer: 1 mark Other answer: 0 mark

Answer: A---------r

B--------q

C---------p

D---------s

2. **What do you mean by Asymptotic Analysis? Define different Asymptotic Notations required for analysis of Algorithm and explain with suitable examples. [5]**

Correct meaning of Asymptotic Analysis: 1 mark

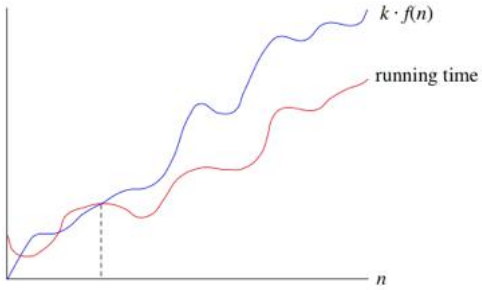
Defining of Asymptotic Notations: 4marks

Partial correct: 1 to 2.5 marks

Answer: Asymptotic analysis of an algorithm refers to defining the mathematical boundation/framing of its run-time performance.

Big-O

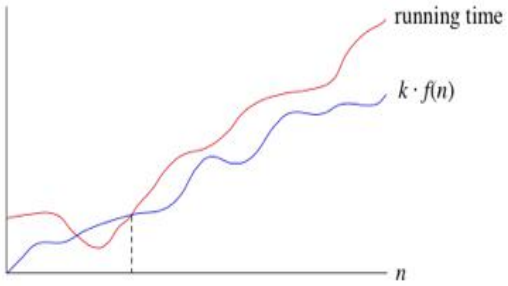
Big-O, is an Asymptotic Notation for the worst case, or ceiling of growth for a given function. It provides us with an asymptotic upper bound for the growth rate of runtime of an algorithm. Say f(n) is your algorithm runtime, and g(n) is an arbitrary time complexity you are trying to relate to your algorithm. f(n) is O(g(n)), if for some real constants c (c > 0) and n0, f(n) <= c g(n) for every input size n (n > n0).



Big-Omega

Big-Omega, is an Asymptotic Notation for the best case, or a floor growth rate for a given function. It provides us with an asymptotic lower bound for the growth rate of runtime of an algorithm.

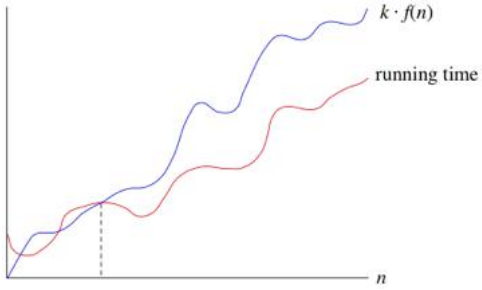
f(n) is Ω(g(n)), if for some real constants c (c > 0) and n0 (n0 > 0), f(n) is >= c g(n) for every input size n (n >n0).



Asymptotic analysis of an algorithm refers to defining the mathematical boundation/framing of its run-time performance.

Big-O

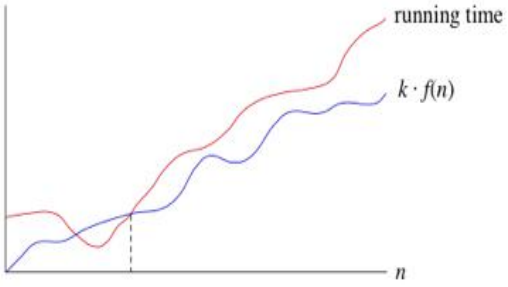
Big-O, is an Asymptotic Notation for the worst case, or ceiling of growth for a given function. It provides us with an asymptotic upper bound for the growth rate of runtime of an algorithm. Say f(n) is your algorithm runtime, and g(n) is an arbitrary time complexity you are trying to relate to your algorithm. f(n) is O(g(n)), if for some real constants c (c > 0) and n0, f(n) <= c g(n) for every input size n (n > n0).



Big-Omega

Big-Omega, is an Asymptotic Notation for the best case, or a floor growth rate for a given function. It provides us with an asymptotic lower bound for the growth rate of runtime of an algorithm.

f(n) is Ω(g(n)), if for some real constants c (c > 0) and n0 (n0 > 0), f(n) is >= c g(n) for every input size n (n > n0).

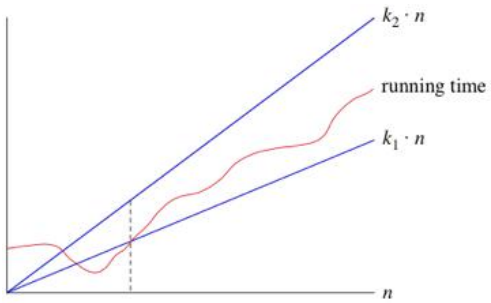


Theta

Theta, is an Asymptotic Notation to denote the asymptotically tight bound on the growth rate of runtime of an algorithm.

f(n) is Θ(g(n)), if for some real constants c1, c2 and n0 (c1 > 0, c2 > 0, n0 > 0), c1 g(n) is < f(n) is < c2 g(n) for every input size n (n > n0).

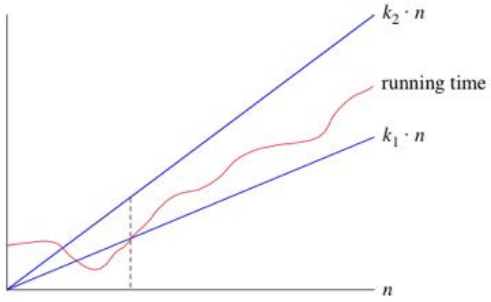
∴ f(n) is Θ(g(n)) implies f(n) is O(g(n)) as well as f(n) is Ω(g(n)).



Theta, is an Asymptotic Notation to denote the asymptotically tight bound on the growth rate of runtime of an algorithm.

f(n) is Θ(g(n)), if for some real constants c1, c2 and n0 (c1 > 0, c2 > 0, n0 > 0), c1 g(n) is < f(n) is < c2 g(n) for every input size n (n > n0).

∴ f(n) is Θ(g(n)) implies f(n) is O(g(n)) as well as f(n) is Ω(g(n)).



3. Write the Algorithm for Insertion Sort and derive its best and worst case time complexities. Explain the Insertion Sort Algorithm with the following data. [5]

<12, 6, 2, 5 ,11, 4, 8, 1>

Correct algorithm:2marks

Best worst case time complexity& Steps with given data: 3marks

Partial correct algorithm and working on data set : 1 to 3 marks

Anwer:

Insertion Sort(A)

1. for i = 2 to length(A)
2. x=A[i]
3. j=i-1
4. while j>0 and A[j]>x
5. A[j+1]=A[j]
6. j = j-1
7. end while
8. A[j+1]=x

Best case :
1. Outer for loop runs for (n-1) times
2. Inner while loop exits the loop in the very first comparison
3.1+1+1+...+(n-1) times =>O(n)

Worst case :
1. Outer for loop runs for (n-1) times
2. In worst case the inner while loop runs for (n-2) times

3.1+2+3+...+(n-1) = ((n-1)*n)/2 => $O(n^2)$

Explaination with dataset:

12 6 2 5 11 4 8 1

6 12 2 5 11 4 8 1
6 2 12 5 11 4 8 1

2 6 12 5 11 4 8 1
2 6 5 12 11 4 8 1
2 5 6 12 11 4 8 1

2 5 6 11 12 4 8 1

2 5 6 11 4 12 8 1
2 5 6 4 11 12 8 1
2 5 4 6 11 12 8 1
2 4 5 6 11 12 8 1

2 4 5 6 11 8 12 1
2 4 5 6 8 11 12 1

2 4 5 6 8 11 1 12

2 4 5 6 8 1 11 12
2 4 5 6 1 8 11 12
2 4 5 1 6 8 11 12
2 4 1 5 6 8 11 12
2 1 4 5 6 8 11 12
1 2 4 5 6 8 11 12

4. Solve the following recurrences. [5]

(i) $T(n) = 3T(n/4) + n^2$

(ii) $T(n) = T(n) = T(\sqrt{n}) + 1$, $T(n)$ is constant for $n \leq 2$

Scheme: Correct solution (i)-2.5 marks

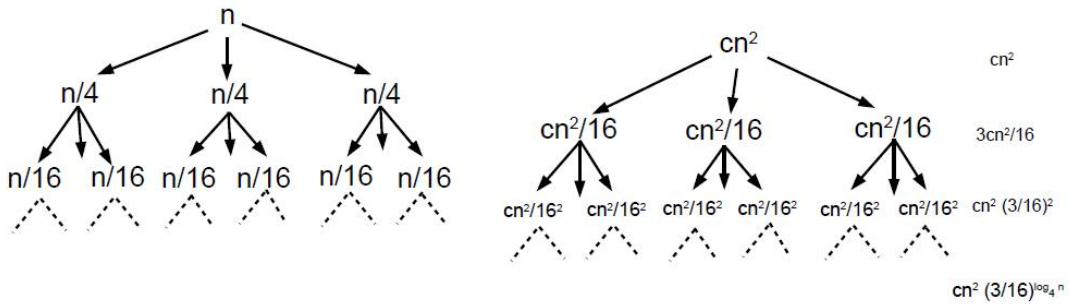
Correct solution (ii)-2.5 marks

Partial correct solution-1 to 2.5 marks

Answer: (i) $T(n) = 3T(n/4) + n^2$

This can be solved by master theorem or recurrence tree method:

Note that the number of levels = $\log_4 n + 1$



Also the number of leaves = $3\log_4 n = n\log_4 3$

The total cost taken is the sum of the cost spent at all leaves and the cost spent at each sub-

division operation. Therefore, the total time taken is

$$T(n) = \sum_{i=0}^{\log_4(n)-1} cn^2(\tfrac{3}{16})^i + n^{\log_4(3)} \times T(1)$$
$$= \frac{\frac{3}{16}^{\log_4(n)} - 1}{\frac{3}{16} - 1} cn^2 + n^{\log_4(3)} \times T(1)$$
$$= \frac{1 - \frac{3}{16}^{\log_4(n)}}{1 - \frac{3}{16}} cn^2 + n^{\log_4(3)} \times T(1)$$
$$= \frac{1 - n^{\log_4(\frac{3}{16})}}{1 - \frac{3}{16}} cn^2 + n^{\log_4(3)} \times T(1)$$
$$= d'cn^2 + n^{\log_4(3)}T(1) \text{ where the constant } d' = \frac{1 - n^{\log_4(\frac{3}{16})}}{1 - \frac{3}{16}}$$
$$= d'cn^2 + n^{\log_4(3)}T(1). \text{ Therefore, } T(n) = O(n^2)$$

Since the root of the computation tree contains cn2, $T(n) = \Omega(n^2)$. Therefore, $T(n) = \theta(n^2)$

(ii) T(n) = T(n) = T(√n) + 1 , T(n) is constant for n ≤ 2

Introduce a change of variable by letting $n = 2^m$.

$\Rightarrow T(2^m) = 2 \times T(\sqrt{2^m}) + 1$

$\Rightarrow T(2^m) = 2 \times T(2^{m/2}) + 1$

Let us introduce another change by letting $S(m) = T(2^m)$

$\Rightarrow S(m) = 2 \times S(m/2) + 1$

$\Rightarrow S(m) = 2 \times (2 \times S(m/4) + 1) + 1$

$\Rightarrow S(m) = 2^2 \times S(m/2^2) + 2 + 1$

By substituting further,

$\Rightarrow S(m) = 2^k \times S(m/2^k) + 2^{k-1} + 2^{k-2} + ... + 2 + 1$

To simplify the expression, assume $m = 2^k$

$\Rightarrow S(m/2^k) = S(1) = T(2)$. Since $T(n)$ denote the number of comparisons

We now have, $S(m) = T(2^m) = m+2$. Thus, we get $T(n) = m+2$, Since $m = \log n$, $T(n) = \log n+2$
Therefore, $T(n) = \theta(\log n)$

5. What is Divide-and-Conquer approach? Write the Algorithm for Binary Search and show how Divide-and-Conquer approach is applied in Binary Search. [5]

Scheme: Divide and conquere approach: 2 marks

Binary search algorithm and approach: 3 marks

Answer:

Divide problem into several smaller subproblems

Normally, the subproblems are similar to the original

Conquer the subproblems by solving them recursively

Base case: solve small enough problems by brute force

Combine the solutions to get a solution to the subproblems

And finally a solution to the orginal problem

Divide and Conquer algorithms are normally recursive

```
binarysearch(number n, index low, index high, const keytype S[], keytype x)
        if low ≤ high then
          mid = (low + high) / 2
          if x = S[mid] then
            return mid
          Else if x < s[mid] then
            return binarysearch(n, low, mid-1, S, x)
          else
```

```
        return binarysearch(n, mid+1, high, S, x)
    else
        return 0
end binarysearch
```

- Divide: entire dataset is divided from middle index. search lower or upper half
- Conquer: search selected half
- Combine: None

It is possible to take greater advantage of the ordered list if we are clever with our comparisons. In the sequential search, when we compare against the first item, there are at most n−1n−1 more items to look through if the first item is not what we are looking for. Instead of searching the list in sequence, a binary search will start by examining the middle item. If that item is the one we are searching for, we are done. If it is not the correct item, we can use the ordered nature of the list to eliminate half of the remaining items. If the item we are searching for is greater than the middle item, we know that the entire lower half of the list as well as the middle item can be eliminated from further consideration. The item, if it is in the list, must be in the upper half.

We can then repeat the process with the upper half. Start at the middle item and compare it against what we are looking for. Again, we either find it or split the list in half, therefore eliminating another large part of our possible search space.

6. W**e are given two arrays (X & Y), the array X with m elements arranged ascending order and the array Y with n elements arranged in descending order. Write an Algorithm to merge these two arrays (X & Y) into a third array Z in ascending order.** [5]

Scheme : Correct algorithm:5 marks

Partialy correct solution: 1 to 3 marks

Answer:

```
Mergeing(int X[]. int Y[], int m, int n)
                { int Z[30],i,j;
                i=0;j=n-1;
                while   i < m and j >=0
                      if (X[i] <=Y[j])
                Z[k]=X[i];
                         i++;k++;
                else  Z[k]=Y[j];
```

```
                k++;j--;
            while  i < m
                Z[k]=X[i];
                i++;k++;
            while j >=0
                Z[k]=Y[j];
                j--;k++;
print("Merging ",Z)}
```