what is an Algorithm

write an algorithm to add two integers.

1. declare a,b and c as integers.
2. get the values of a and b.
3. compute c=a+b
4. print the value of c.

---

write an algorithm to find the biggest of three integers.

1. declare a,b,c as integers.
2. get the values of a, b and c.
3. check if a > b and a > c, then print a is the greatest.
4. else check if b > a and b >c, then print b is the greatest
5. else print c is the greatest

---

write an algorithm to add addition of n integers.

1. declare n, sum=0 as integer.
2. get the value of n.
3. declare an array a [n] as integer.
4. do i=0 to i<n
5. get the array a[n] values.
6. do i=0 to i<n
7. sum=sum+a[i]
8. print the value of sum.

Difference between theoritical and practical running time.

---

frequency count method

1. Identify the number executable and non-executable statements present in the algorithm.
   Executable: assignment, conditional, looping, looping, branching
   Non-Executable: declarations, input, output, function prototype, pre-processor, comment, inbuilt
2. Identify the frequency of each executable statements. (Frequency- How many times will an executable statement be executed)
3. Add the frequency of each executable statements.

example
int a; // non exe
int a=10; // exe (exe means compiler will have to allocate some memory)

`

example1 of frequency count method
sum=sum+10; // frequency is 1
example2
if (a<b) // frequency is 1
sum=sum+10 // this step may happen or not (frequency is 0 or 1)
// total frequency is 2.

example 3
for(i=0;i<n;i++) // 1, n+1, n times (from 0 to n-1, and once i will be equal to n)
sum=sum+10;
total frequency: 3n+2

example 4
void main() {
int a[100], i, n, sum=0; // e-1 assignment only
clrscr();
print(“Enter the value of n”);
scanf("%d", &n);
printf(“Enter the array values”);
for(i=0;i<n;i++) // e-1, n+1, n

scanf("%d",&a[i]); // n [we won't ignore scanf since it comes under loop]
for(i=0;i<n;i++); // e-1,n+1,n
sum=sum+a[i] // e-n
printf("The addition of n integers=%d", sum);
getch();
}

Total frequency: 6n+5

example 5
for(i=0;i<n;i ++) // e - 1,n+1, n = 2n+2
for(j=0;j<n;j++) // e - (2n+2) * n = e -n, (n^2 + n) , n^2
sum=sum+10; // n^2

`

> in frequency count method we have to always consider the max frequency.
> When we talk about the time complexity, we say n times, not 6n times etc, because it is considered linear, we neglect the other terms.

-------- 22 nd September ----------

What is the name of first virus program

brain (made by baisd and amjid)

Frequency count method:

avoid unnecessary loops. Time complexity grows with the increase of nesting of loops.
one loop - n
two loop nested - n^2
three loop nested - n^3
Constants are not important, because the biggest power is important, powers less than it

won't matter, $3n^2 + n^2 = n^2$

Asymptotic notations:

big-oh(O): any function f(n) can be denoted as f(n)= O(g(n)),
this means that O(g(n)) denotes upper bound of f(n).

Write an algorithm to find the given integer is present in the given unsorted array or not.

> Example:
> input: 10, 35, 2, 22, 100
> sn=200
> output: not present
> // 5 comparisons are needed, you cannot do n-1 comparisons, and say its not present
> Example:
> input: 10, 35, 2, 22, 100
> sn=10
> output: present
> // only takes 1 comparison.
> // time depends on comparison.
> // so n we can say that n is the biggest time notation.

Omega(Ω): any function f(n) can be denoted as f(n)= Ω(g(n)),
this means that Ω(g(n)) denotes lower bound of f(n).

Theta(θ): any function f(n) can be denoted as f(n)= θ(g(n)),
this means that θ(g(n)) denotes lower bound of f(n).

> Example: write an algorithm to find the biggest number of given unsorted array.
> input: 10, 35, 2, 22, 100
> output: 100
> minimum time: 5 comparisons (even if biggest number is present at first place. we still need to check all values)

> maximum time: 5 comparisons
> so we can not find the minimum time, so theta doesn't exist.

```
when no minimum and max time exists, theta doesn't exist
```

Arrange asymptotic notations in order:

1. big-oh
   (because ex- pick best algorithms among 3, in worst case how much time will it take)
   (most books use big-oh in analysis)
2. omega
   (if omega fails, then go to next one, do not depend too much on omega)
3. growth rate analysis
4. space complexity analysis

Thetha can not be always used.

$6n + 5$
$3n^2 + 4n + 2$
same the same problem in n time ,
we can reduce the loops to reduce the time complexity.

Growth Rate Analysis.

n denotes the number of inputs
with the increase in inputs, how does an algorithm changes in growth rate.
https://docs.google.com/drawings/d/15ErubQZYNa9XihNPST-BGkPkF4LhV_vewKis2-S1qrE/edit?usp=sharing

How to improve the time complexity? Like moving from
cubic to quadric (fine)
quad to linear (good)
linear to log (very good)
log to 1 (best)

why is it important to improve the time complexity?
eg. biometric system takes 1 hour to identify? would it be good?

---

27 Sept

Time Complexity

---

write two different algorithm for sorting and analyze its time complexity.
Bubble sort:
n=5
input: 34,12,45,9,1
output: 1, 9,12,34,45
total number of iterations=n-1
total number of comparisons in all the iterations=n-1+n-2+...+n-(n-1)

iteration1: (Four iterations = n-1 comparisons)
first comp: 12,34,45,9,1
second comp: 12,34,45,9,1
third comp: 12,34,9,45,1
fourth comp: 12, 34,9,1,45

iteration2: (n-2 comparisons)
iteration3: n-3
iteration4: n-(n-1)
total number of comparisons = 4+3+2+1=10

insertion sort:
n=5
input: 34,12,45,9,1
output: 1,9,12,34,45
number of iterations=n-1
number of comparisons in all the iternations= n - (n-1) + n-(n-2) + ... + n-1
iteration 1:

1st comp: 12,34,45,9,1
iteration 2:
1st comp: 12,34,45,9,1
2nd comp: 12,34,45,9,1
iteration3:
1st comp: 9,12,34,45,1
iteration4:
1st comp: 1,9,12,34,45

Total comparisons = 5
Insertion sort is better when data is partially sorted.
best time complexity of insertion sort is n.
(best case will be when data is in decreasing order)
worst case time complexity of insertion is n.

Hmw: find at least 2 algorithm which fall in the columns of table

O(1)	**O(log n)**	**O(n)**	**O(n log n)**	**O(n^2)**	**O(n^3)**	**O(2^n)**
Linear Search	B	Bubble Sort	Merge Sort	Selection Sort	B	A
Binary Search	B	Insert Sort	Quick Sort	A	B	A
A	B	Tim Sort	Heap Sort	A	B	A
A	B	Shell Sort	B	A	B	A

O(1)- finding the smallest or largest in the sorted array
O(1)- push,pop
O(log n) - binary search, insertion in bst,
O(n) - array traversal, linear search, finding max or min in unsorted
O(n log n) - Merge Sort, Heap Sort, Quick sort
O(n^2) - selection sort, finding duplicates, bubble sort

O(n^3) - matrix multiplication, graph traversals
O(2^n) - finding all subsets of given set, towers of hanoi

Time Complexity of If and Else
if(conditon){
1
2
...
n
}
else{
1
2
...
n
}

whenever asked to calculate the time complexity , Big oh is generally calculated (Worst Case) (If not specified).

Homework Problems:

1. find the time complexity of the following program segments
   (we need to find frequency counter)

```
for(i=0;i<n;i++){
    sum=sum+a[i];
    for(j=0;j<n;j++) {
        flag=flag-1;
    } }

for(i=0;i<n;i++){
    for(j=0;j<m;j++){
        sum=sum+a[i];
    }
}

while(i<n) {
```

```
    if(a[i] > a[j]) {
        temp=a[i];
        a[i] = a[j];
        a[j] = a[temp];
    }
}

while ( (i<n) || (j<m) ) {
    while(i<n){
        temp = a[i];
        i++;
    }
    while(j<m) {
        temp = a[j];
        j++;
    }
}
```

Q- The biggest problem in algorithm analysis?
Solution verification

- **proof by induction** (test that it is giving right answer for small values, assume your algorithm is true upto some limit k, then prove for k+1)
- **proof by contradiction** (assume that it is wrong, then prove that it is true)

Example:
Fibonnaci Series
P(0), P(1), P(2) + ... + P(n+1)

Proof by induction:

fn = fn-1 + fn-2, fi<(5/3)^i
i>=1
i=1, (5/3)^1 ,^ 1<(5/3)
i=2, $(5/3)^2$ 2<(25/9)
The theorem is true upto some limit k
Now proving that theorem is true for k+1,
f(k+1) < $(5/3)^{k+1}$
f(k+1) < f(k) + f(k-1)

$< (5/3)^k + (5/3)^{k-1}$
$< (5/3)^{k-1} * (5/3) + (5/3)^{k-1}$
$< (5/3)^{k-1} (5/3 + 1)$
$k+1 < (5/3)^{k-1} (8/3)$

//messed up, proof not in syllabus?

> CAT 1 EXPECTED QUESTIONS
> what kind of question will come from module 1 in CAT 1
>
> 1. we will be given algorithms and will have to find time complexities.

> two three algos will be given, find the time complexity based on Big oh, and arrange them in order, first fastest, second fatest, third fastest

> find algos and time complex will be given, you have to find growth rate, find the response time, and make a graph of Growth Rate Analysis. (link)[https://prnt.sc/1u7j7bp]

> worst case and algo given, find the best case, or reduce the time complexity in the best case.
> reducing time complexity in the best case

> only use frequency count method
> polynomial no need to specify?

> Explain asymptot notations with examples (Theory Question will be rare)

```
Evaluation will be very strict in this subject
```

cat will be till priority queues (everything teaches till thursday)

Lab
Next week on wednesday, viva will be taken,
marks will be given,
go through all the algorithms, stack algorithms, etc.
question will be asked from program (Lab cycle sheets, stack data structures)