position is

{4c, 4b, 4d, 4a}. So, the next iteration will be on {4c, 4b, 4d} and {4a}. Clearly, no matter what we do on the first sub array, 4a is already out of the original order as it will stay on the right of 4b and 4c, and hence Quick sort is unstable.

Question 4: (continued)

c. You would like to determine which of your Facebook friends are early adopters. So, you have decided to sort them using Facebook account ids which are 64-bit numbers. Which sorting algorithm will be most appropriate – **Insertion sort, Merge sort, Quicksort, Counting sort or Radix sort? Explain why.**

Ans. Radix sort as a 64 bit number can be represented as 4 digits of 16 bit numbers. So, we can use Radix sort with 4 passes to do the sort in linear time. Other algorithms will take O(nlgn) or O(n2).

b. (i) 2 points - Explain with an example what is meant by "Mathematics is not sound".

Ans. "Sound" means only True statements can be proved. For example, A OR not A is always true for a binary value of A. However, A and Not A is always false. But we can also prove this false statement as shown below:

For A = 0, not A is 1. Also, for A =1, not A is 0. So, their logical "and" is always False. Thus, we can prove False statements in Logic. Thus, Math is NOT sound.

(ii) **3 points** - Show how Quicksort is not stable by using in-place random partitioning algorithm and the following 4 numbers {4a, 4b, 4c, 4d} (show all steps).

Ans. Consider 4d as the pivot. Initially i is at 4a and j is at 4c. They are both stuck as their value is 4 (duplicate). So, we swap them and then advance i to the right and j to the left. Thus, we have {4c, 4b, 4a, 4d}

Pivots can be made good pivots by excluding all bad pivots – e.g. this can be done by using pivot call as a waisted self call without doing the real recursion but just calling the recursion with the same parameter value etc (see QuickSelect slides). It can be shown that this way we achieve partition at O(n) as the recursion tree height is lgn instead of n. Hence, running time is
O(nlgn) instead of O(n.n).

b. (i) 2 points - Explain with an example what is meant by "Mathematics is not sound".

Ans. "Sound" means only True statements can be proved. For example, A OR not A is always true for a binary value of A. However, A and Not A is always false. But we can also prove this false statement as shown below:

For A = 0, not A is 1. Also, for A =1, not A is 0. So, their logical "and" is always False. Thus, we can prove False statements in Logic. Thus, Math is NOT sound.

(ii) **3 points** - Show how Quicksort is not stable by using in-place random partitioning algorithm and the following 4 numbers {4a, 4b, 4c, 4d} (show all steps)

c. Use RadixSort, with two bucket arrays and radix = 12, to sort the following array: [63, 1, 48, 53, 24, 10, 12, 30, 100, 141, 17]. Show all steps of the sorting procedure. Then explain why the running time is O(n). What would be the running time if you used ONE bucket? What would be the running time if you used ONE bucket?

Ans.

Here the range is large and one bucket will not work. We need to use 2 buckets as with 2 buckets, the max number we can represent is

143 = 12.q1 + r using the max value of q1 = r is 11. The max number in the array is 141. So, 2 buckets will suffice.

c. Use RadixSort, with two bucket arrays and radix = 12, to sort the following array: [63, 1, 48, 53, 24, 10, 12, 30, 100, 141, 17]. Show all steps of the sorting procedure. Then explain why the running time is $O(n)$. What would be the running time if you used ONE bucket? What would be the running time if you used ONE bucket?

Ans.

Here the range is large and one bucket will not work. We need to use 2 buckets as with 2 buckets, the max number we can represent is

143 = 12.q1 + r using the max value of q1 = r is 11. The max number in the array is 141. So, 2 buckets will suffice.

The formulae we need to use are as follows:

1. Bucket r - use input values x and remainder using mod 12.

2. Bucket q1 needs to be filled by reading values (left to right) from bucket r but divided by 12 and and taking the quotient.

If one bucket is used, then we would need n^2 size as the range is close to 12^2 where 12 is the number of inputs. So, the time would be O(n^2).

Question 4: 13 points (4 + 5 + 4)

a. What is the worst case running time of Quicksort? Can you improve the worst case running time of quicksort? If so, describe to what value and how.

Ans. Worst case running time of Quick Sort (QS) is $\Omega(n^2)$. Yes, it can be improved to O(nlgn) if we can ensure that all the pivots are good pivots.

Pivots can be made good pivots by excluding all bad pivots – e.g. this can be done by using pivot call as a waisted self call without doing the real recursion but just calling the recursion with the same parameter value etc (see QuickSelect slides). It can be shown that this way we achieve partition at O(n) as the recursion tree height is lgn instead of n. Hence, running time is
O(nlgn) instead of O(n.n).

Question 3: 11 points (4 + 4 + 3)

a. (a) Assume you are creating an array data structure that has a fixed size of n. You want to backup this array after every so many insertion operations. Unfortunately, the backup operation is quite expensive, it takes n time to do the backup. Insertions without a backup just take 1 time unit.

(i) How frequently can you do a backup and still guarantee that the amortized cost of insertion is O(1)?

Ans: You can backup the array after every n insertions. Total cost with n insertion is n. To back up after n insertions is n. So, total for a backup after each n insertion is 2n = O(n) for n insertions. Hence, per instruction, the amortized cost is O(1) (see answer for (ii) below).

(ii) Prove that you can do backups in O(1) amortized time. Use the accounting method for your proof.

array after every so many insertion operations. Unfortunately, the backup operation is quite expensive, it takes n time to do the backup. Insertions without a backup just take 1 time unit.

(i) How frequently can you do a backup and still guarantee that the amortized cost of insertion is O(1)?

Ans: You can backup the array after every n insertions. Total cost with n insertion is n. To back up after n insertions is n. So, total for a backup after each n insertion is 2n = O(n) for n insertions. Hence, per instruction, the amortized cost is O(1) (see answer for (ii) below).

(ii) Prove that you can do backups in O(1) amortized time. Use the accounting method for your proof.

Ans. We have $C_{insertion} = 1$ and $C_{backup} = n$. Let's use $C^{\wedge}_{insertion} = 2$ and $C^{\wedge}_{backup} = 0$. So, for n insertions, we have a profit of n as for each insertion, we make 2-1 = 1 Cyber Dollar. To back up will need n. So, we have enough to pay for the backup. The amortized cost, C^ for n insertion is 2n which is O(n). So, C^ per insertion is

$O(n) / n = O(1)$.

(b) Use the QuickSelect algorithm to manually compute the 5th smallest element of the array [1, 5, 23,

0, 8, 4, 33]. Assume that the rightmost element is used as the pivot in each case. Show what happens in each self-call, indicating the new input array and the current value of *k*.

Ans. QS (1,5, 23, 0, 8, 4, 33), k = 5

L = [1,5,23,0,8,4], E = [33], G = [ ]

k = 5 <= |L|

QS (1,5,23,0,8,4), k = 5

L = [1, 0], E = [4], G = [5, 23, 8]

k' = k - |L| - |E| = 5 - 2 - 1 = 2

QS (5, 23, 8), k = 2

L = [5], E = [8], G = [23]

|L| < k ≤ |L| + |E|, so **return 8**

b. Use **Iterative method**

$$T(n) = T\left(\frac{n}{2}\right) + T(n-2) + c \quad for\ n > 0$$ [Use Iterative method

T(n) = 1 (for n = 0 -> n =1

Correction -

T(n) = 1 (for n = 0 -> n =1

Correction -

$$\begin{cases} T(n) = 3T(n-1) + 1 \\ T(1) = 0 \end{cases}$$

Ans. Using iterative method, we have,

T(n) = 3{3T(n-2) +1} + 1

= 3^2 T(n-2) + 3 + 1

= 3^2 { 3T(n-3) + 1} + 3 + 1

= 3^3 T(n-3) + 3^2 + 3 + 1

…

…

= 3^k T(n-k) + 3^(k-1) + 3^(k-2) + ……. 3^1 + 3^0

Setting n = k + 1, and using T(0) = 0, we get,

T(n) = 1 + 3 + 3^2 + ……3^(n-2). Using sum series formula, we have,

T(n) = [3^(n-1) – 1] / (3-1) = [3^(n-1) – 1] / 2.

...

= 3^k T(n-k) + 3^(k-1) + 3^(k-2) + ... 3^1 + 3^0

Setting n = k + 1, and using T(0) = 0, we get,

T(n) = 1 + 3 + 3^2 + ...3^(n-2). Using sum series formula, we have,

T(n) = [3^(n-1) – 1] / (3-1) = [3^(n-1) – 1] / 2.

Question 2: (continued)

c. Use Induction to show that

$$\sum_{r=1}^{n} r(r+1) = \frac{1}{3}n(n+1)(n+2)$$

Ans.

Base case r = i = 1. Left side = 1(1+1) = 2; Right side = (1/3).1(1+1)(1+2) = 6/3 = 2. Hence base case is proved.

Inductive case: Assume for k and show for k+1. So, we have (sum over r = 1 to k) ∑ r(r+1) =

$$(2^n + n^2)(n^3 + 3^n).$$

Ans.

We can apply the product rule i.e. the big O of the product will be the BigO of each term under the parenthesis. Since $2^n >> n^2$, the first term is $O(2^n)$ and the same way, the 2nd term is $O(3^n)$. Hence the function is $O(2^n \cdot 3^n)$ which is $O(6^n)$.

Question 2: 12 points (4 + 4 + 4)

For each of the following recurrences, derive an expression for the running time using iterative

substitution of Master Theorem.

a. Consider the following recurrence algorithm **[Use Master Theorem – See LAST Page]**

```
long power(long x, long n)
  if (n==0) return 1;
  if (n==1) return x;
  if ((n % 2) == 0)
    return power(x*x, n/2);
  else
    return power(x*x, n/2) * x;
```

Assume n is power of 2.

i. (2 points) Write a recurrence equation for T(n)

Soln.

T(0) = 2
T(1) = 2
T(n) = T(n/2) + 6

Assume n is power of 2.

i. (2 points) Write a recurrence equation for T(n)

Soln.

T(0) = 2
T(1) = 2
T(n) = T(n/2) + 6

ii. (2 points) Solve recurrence equation using Master's method i.e. give an expression for the runtime T(n).

Soln.

A =1, b = 2, c = 6, k = 0.
So, $a = b^k$
Thus, we have, T(n) = Theta (lgn)

b. Use **Iterative method**

}

Ans. Outer for loop is n + 100 times, first inner for loop is n^2 times and 2^{nd} inner for loop is n + n = 2n times.

Hence, total running time T(n) = O (n^3) + O(n^2) = O(n^3)

b. Determine whether f is O(g), Theta (g) or Omega (g). Mention all that applies. Show your work.

i) $f = 2^{2n}$, $g = 2^n$

Lt f/g (with n -> infinity) = 2^{2n} / 2^n = 2^n = Infinity. So, f is Omega (g) i.e. g is o(f) as shown below:

Lt g/f = 2^n / 2^{2n} = $1/2^n$ = 1/Infinity = 0.

ii) $f = n^{\lg m}$ and $g = m^{\lg n}$; Show your reasoning / work.

shown below:

Lt g/f = 2^n / 2^{2n} = $1/2^n$ = 1/Infinity = 0.

ii) f = $n^{\lg m}$ and g = $m^{\lg n}$; Show your reasoning / work.

Ans. Lt (n -> inf) f/g -> Lt (n -> inf) f' / g' (Using L'Hopital rule) -> c.(lg m).n^ (lg m-1). /(1/n (m^lg n))

which is c.lg(m) . n^lg m / m^lg n - same as the c.lg(m) f/g. Thus, if we continue taking L'Hopital rule, we will get $(c\lg(m))^k$. f/g after doing it for k times. This, this approach will not yield a finite number.

Thus, we would need to use some other technique. Let's try taking log of f/g as it will not change the results unless f/g is 0 or infinity.

Thus, we have, lg(n^lg m) - lg (m^lg n) which is lg m. lg n - lg n. lg m = 0. Thus, f is Theta of g as f/g = 1 and g/f is also 1.

c. Give a Big O estimate for the following function

b. Other - Master, guess etc **(sample Medium to hard type of question)**

Show that in the recurrence

$$T(n) = \max_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n) ,$$

$$T(n) = \Omega(n^2).$$

Ans. Do it in class

3.

a. Radix, Bucket, Counting

(Soln. to HW6 problem 2) **(Sample Medium to hard type of question)**

Question 1: 10 points (3 + 4 + 3)

a. Show the running time of the following code using big O notation (*show your work*):

```
for (int i = 0; i < n + 100; ++i)
{
      for (int j = 0; j < i * n ; ++j)
      {
            sum = sum + j;
      }

      for (int k = 0; k < n + n + n; ++k)
      {
            c[k] = c[k] + sum;
      }
}
```

Ans. Outer for loop is n + 100 times, first inner for loop is n^2 times and 2^{nd} inner for loop is n + n = 2n times.

Hence, total running time T(n) = O (n^3) + O(n^2) = O(n^3)

b. Determine whether f is O(g), Theta (g) or Omega (g). Mention all that applies. Show

Sample Mid Term for Review

1.

a. Show that $n^2 + 2$ is *not* O(n) [do it in class]

b. Let f(n) = (lg n)^k and g(n) = n^{ϵ} . Find the relationship between f and g consider asymptotic growth **(Sample Medium to hard type of question)** [do it in class]

a. Iteration (s)

$$\begin{cases} T(n) = T(n-1) + 2 \\ T(1) = 1 \end{cases}$$

Ans.

T(n)= T(n-1) +2*1
T(n)= T(n-2) +2*2
T(n)= T(n-3) +2*3
T(n)= T(n-4) +2*4

Generalized recurrence relation at the kth step of the recursion:
T(n)= T(n-k) +2*K

We want T(1). So we let n-k = 1. Solving for k, we get k = n - 1. Now plug back in.

T(n) = T(n-k) + 2*k
T(n) = T(1) + 2*(n-1), and we know T(1) = 1
T(n) = 2n-1
We are done. Right side does not have any T(…)'s. This recurrence relation is now

T(n)= T(n-4) +2*4

Generalized recurrence relation at the kth step of the recursion:
T(n)= T(n-k) +2*K

We want T(1). So we let n-k = 1. Solving for k, we get k = n - 1. Now plug back

T(n) = T(n-k) + 2*k
T(n) = T(1) + 2*(n-1), and we know T(1) = 1
T(n) = 2n-1
We are done. Right side does not have any T(…)'s. This recurrence relation is s
solved in its closed form, and it runs in O(n) time.

Anoher Iteration -

a. $T(n) = 2T(n - 1) + 1, T(0) = 0$ [do in class]

Sample Mid Term for Review

1.

a. Show that $n^2 + 2$ is *not* O(n) [do it in class]

b. Let f(n) = (lg n)^k and g(n) = n^{ϵ} . Find the relationship between f and g consider asymptotic growth **(Sample Medium to hard type of question)** [do it in class]