winter 2017

1.
```
struct node {
        int data;
        struct Node* next;
    }

struct Node* create_LL (struct Node* head, int val)
{
    struct Node* temp = new Node;
        temp->data = val;
        temp->next = NULL;

if (head == NULL)
{
    head = temp;
    return head;
}
else
{
  struct Node * first = head;
  struct Node* prev;
  while (head)
  {
    prev = head;
    head = head->next;
  }
  prev->next = temp;
  return first;
}
}
```

```
void BSTtree(Node* root, struct Node head)
{
  if (root == NULL) return;
  BSTtree(root->left, head);
  head = create_LL(root->data);
  BSTtree(root->right, head);
}
```

2 ay.
```
class queue
{
    private: int A[n];
             int front, rear;
    public : queue()
             {
                front = -1;
                rear = -1;
             }
}
```

by
```
bool search (int a[], int x)
{
    if( rear == front) return false;
    for(int i = 0; i ≤ rear ; i = (i+1) % n)
    {
        if (a[i] == x)
            return true;
    }
    return false;
}
```

3 ay $\sum_{k=1}^{n} \left\{ 1 + \left[\sum_{i=1}^{k-1} \right] + 1 \right\}$

$\sum_{k=1}^{n} \left[2 + (k-1+1) \right]$

$\sum_{k=1}^{n} \left[k+2 \right]$

$\frac{n(n+1)}{2} + 2(n-1+1)$

$\frac{n^2+n}{2} + 2n$

$n^2 + \frac{5}{2}n$

$O(n^2)$

$\frac{1}{2} + 2$

$\frac{1+}{2}$

by $\sum_{i=n/2}^{n} 1$

$= n - \frac{n}{2} + 1$

$= \frac{n}{2} + 1$

$= O(n)$

Q $T(n) = 2T(n-1)+1 \quad n \neq 1;$

$= 1 \quad n = 1$

$T(n) = 2\left[2T(n-2)+1\right] + 1$

$= 2^2 T(n-2) + 2 + 1$ $\quad n-k=1$, $n=1+k$

$= 2^2\left[2T(n-3)+1\right] + 2 + 1$

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

:

$= 2^k T(n-k) + \cdots + 2^1 + 2^0$ $\quad n-k=1$, $n=1+k$

$n=k$ $= 2^k T(1) + \cdots + 2^1 + 2^0$ $\quad$ sub $n=k+1$, $k=n-1$

$= 2^{k-1} + 2^{k-2} + \cdots + 2^1 + 2^0$ $\quad 2^k T(1) + \cdots + 2^1 + 2^0$

$2^k + 2^{k-1} + \cdots + 2^0$

$= 2^k - 1$

$= 2^{n-1} - 1$

$= 2^{n-1+1} - 1$

$= 2^n - 1$

$= O(2^n)$

Spring 2017

1. returns the no of lyt children in a tree

Global variables X

struct TreeNode

{ int data;

TreeNode * lyt, right;

};

int leftCount (TreeNode *ptr)

{

if (ptr == NULL) return 0;

else if (ptr->lyt != NULL)

return 1+leftCount (ptr->lyt) + leftcount +(ptr->right);

else

return LeftCount (ptr->right);

}

Spring 2017

1. returns the no. of left children in a tree

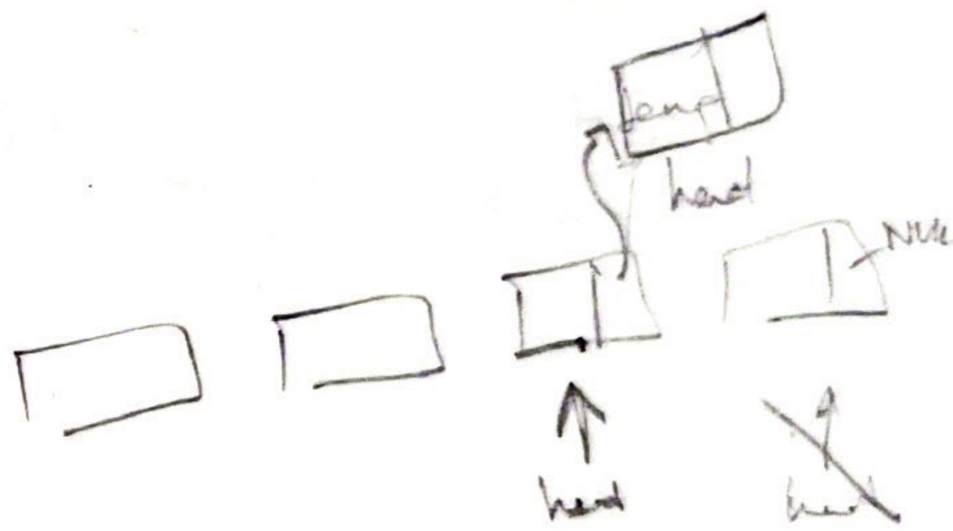
```
Global variables X

struct TreeNode
{
    int data;
    TreeNode * left, right;
};

int leftCount (TreeNode *ptr)
{
    if (ptr == NULL) return 0;
    else if (ptr->left != NULL)
      return 1+leftCount (ptr->left) + leftcount +(ptr->right);
    else
    return LeftCount (ptr->right);
}
```

2. reverses the last 4 nodes of list
   fewer than 5 then entire list should be reversed

```
int length(Node * head)
{
    int n=0;
    while(head)
    {
        n++;
        head = head->next
    }
    return n;
}
Node *reverse(Node* head)
{
    Node *temp;
    Node *prev = NULL;
    while(head != NULL)
    {
        temp = head->next;
        head->next = prev;
        prev = head;
        head = temp;
    }
    return prev;
```



```
void  reverse_last (Node* head, int n)
{
   n=length(head);
   Node *temp=head;
   if(n < 5)
      head  =reverse(head);
   else
   {
     for(i=0; i<n-4-1; i++)
      {
         temp → next;
      }
     temp → next  =reverse (temp → next);
}
```

3. $T(n) = 2T(n/2) + (n-1)$; $n > 1$

$= 0$; $n = 1$

$$T(n) = 2\left[2T\left(\frac{n}{2^2}\right) + \left(\frac{n}{2} - 1\right)\right] + n - 1$$

$$= 2^2 T\left(\frac{n}{2^2}\right) + n - 2 + n - 1$$

$$= 2^2 T\left(\frac{n}{2^2}\right) + 2n - 2 - 1$$

$$= 2^2\left[2T\left(\frac{n}{2^3}\right) + \left(\frac{n}{2^2} - 1\right)\right] + 2n - 2 - 1$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + n - 2^2 + 2n - 2 - 1$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 3n - 2^2 - 2^1 - 2^0$$

$$\vdots$$

$$= 2^k T\left(\frac{n}{2^k}\right) + kn - \left(2^{k-1} + 2^{k-2} + \cdots + 2^1 + 2^0\right)$$

Sub $n = 2^k$

$$= 2^k T(1) + k2^k - \left(\sum_{i=0}^{k-1} 2^i\right)$$

$k = \log_2 n$

$$= k \cdot 2^k - \left(2^k - 1\right)$$

$$= \log_2 n \, 2^{\log_2 n} - 2^k + 1$$

$$= n \log_2 n - 2^{\log_2 n} + 1$$

$$= n \log n - n + 1$$

Fall 2016

1.

```
String hash_table[n];
int hash_table_size = n;

int insert(int x)
{
  int index = h(x);   // given h(x) = x % n

  repeat
  {while(hash_table[index] != 0 && hash_table[index] ≤ n)

  if(hash_table[index] == 0)
  {
    hash_table[index] = val;
    return index;
  }
  index = (index + index) % hash_table_size;
  }
  return -1;

}
```

```
int search_value (int x)
{
  int index = h(x) % n

    if (hashtable[index] == x)
      return index

    temp = (index + 1 * 1) % n
      int i = 1
//iterate until element is found or any unoccupied
    while(hashtable[temp] != x && hashtable[temp] != 0)
    {   i = i + 1
        temp = (index + i * i) % n;
    }
    if (hashtable[temp] == x)
    {   return temp;
    }
    else
        return -1
}
```

2. void recursive (Node* head)

```
{
    if (head == NULL || head -> next == NULL)
    return 0; head;

    else
    {
      while (ptr -> next != NULL)
      {
          recursive (ptr -> next);
      }
      cout
```

```
Node* reverse (Node* head)
{
   if(head == NULL || head -> next == NULL)
     return head;

   Node* rest = reverse (head -> next);
   head -> next -> next = head;

   head -> next = NULL;

   return rest;
}
```

```
void print()
{
   struct Node* temp = head;
   while (temp != NULL)
   {
      cout << temp -> data << " ";
      temp = temp -> next;
   }
}
```

```
void reverse (Node *ptr)
{
   if (ptr == NULL)
     return 0;

   else
     reverse (ptr -> next);

   cout << ptr -> data;

}
```

3. $T(n) = 2T(n/2) + 3n \quad ; n > 1 \qquad n = 2^k$

$= 1 \quad ; n = 1$

$$T(n) = 2\left[2T\left(\frac{n}{2^2}\right) + 3\left(\frac{n}{2}\right)\right] + 3n$$

$$= 2^2 T\left(\frac{n}{2^2}\right) + 3n + 3n$$

$$= 2^2\left[2T\left(\frac{n}{2^3}\right) + 3\left(\frac{n}{2^2}\right)\right] + 2(3n)$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 3(3n)$$

$$\vdots$$

$$= 2^k T\left(\frac{n}{2^k}\right) + k(3n)$$

When $n = 2^k$ $\qquad k = \log_2 n$

$$= 2^k T(1) + k(3 2^k)$$

$$= 2^k + 3k 2^k$$

$$= 2^{\log_2 n} + 3(\log_2 n) \cdot 2^{\log_2 n}$$

$$= n + 3 \log_2 n \cdot n$$

$$\boxed{T(n) = n + 3n \log n}$$

3. $T(n) = 2T(n/2) + 3n \quad ; n > 1 \qquad n = 2^k$

$= 1 \quad ; n = 1$

$$T(n) = 2\left[2T\left(\frac{n}{2^2}\right) + 3\left(\frac{n}{2}\right)\right] + 3n$$

$$= 2^2 T\left(\frac{n}{2^2}\right) + 3n + 3n$$

$$= 2^2\left[2T\left(\frac{n}{2^3}\right) + 3\left(\frac{n}{2^2}\right)\right] + 2(3n)$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 3(3n)$$

$$\vdots$$

$$= 2^k T\left(\frac{n}{2^k}\right) + k(3n)$$

when $n = 2^k \qquad k = \log_2 n$

$$= 2^k T(1) + k(3 \cdot 2^k)$$

$$= 2^k + 3k2^k$$

$$= 2^{\log_2 n} + 3(\log_2 n) \cdot 2^{\log_2 n}$$

$$= n + 3 \log_2 n \cdot n$$

$$\boxed{T(n) = n + 3n \log n}$$

Spring 2016

1.

```
bool even_nodes(struct node* node)
{
    if(node == NULL)
    return true;

    int count=0;
    count = count+1
    even_nodes(node->left);
    even_nodes(node->right);
    if(count %2 ==0) return true
    else return false;
}
```

```
if(even(ptr->left) == even(ptr->right))
    return true/false;
else
    return false;
```

2.

```
Void merge_sort(int A[],int B[], int n, int m)
{
   int i=0, j=0, k=0, c[n+m];

   for(i=0; i<n; i++)
   {
      for(j=0; j<m; j++)
```

```
void merge(int A[], int B[], int c[], int x, int y, int z)
{
   int i, j, k;
   int n1 = x;
   int n2 = y;

   i = 0; // for A[]
   j = 0; // for B[]
   k = 0; // for c[]

   while(i<n1 && j<n2)
   {
      if(A[i] <= B[j])
         c[k++] = A[i++];
      else
         c[k++] = B[j++];
   }
   while(i<n1)
      c[k++] = A[i++]

   while(j<n2)
      c[k++] = B[j++]
}
```

3.
```
class Graph
{
void int V;
     list<int> *adj;

     void DFSUtil(int v, bool visited[]);

     public:
       Graph(int v);
       void addEdge(int v, int w);
       void DFS(int v);

};
Graph::Graph(int v)
{  this -> V = V;
   adj = new list<int> [V];
}
void Graph::addEdge(int v, int w)
{  adj[v].push_back(w);
}
void Graph::DFSUtil(int v, bool visited[])
{
   visited[v] = true;
   cout<<v<<" ";  // may be sigle_ll(v);
   list<int>::iterator i;
   for(i = adj[v].begin(); i != adj[v].end(); ++i)
      if(!visited[*i])
         DFSUtil(*i, visited);
  }
void Graph::DFS(int v)
{
   bool *visited = new bool[V];
   for(int i = 0; i<V; i++)   visited[i] = false;
```

winter 2016

10.

	search	delete	insert
UA	O(n) O(n)	O(n) O(n)	O(n) O(n)
SA	O(log n) O(log n)	O(n) O(n)	O(n) O(n)
		O(1) O(1)	O(1) O(1)
DL	O(n) O(n)	O(log n) O(n)	O(log n) O(n)
BST	O(log n) O(n)	O(1) O(n)	O(1) O(n)
	O(1) O(n)		
HT			

a/ struct Node {
int data;
struct Node* link;
};

struct Node* top = NULL;

b/ void push(int data)
{
struct Node* temp;
temp = new Node();
if (!temp)
{ cout << "\n Heap overflow";
exit(1);
}
temp->data = data;
temp->link = top;
top = temp;
}

c/ POP
int ~~isEmpty~~()
{
struct Node* temp;
if (top == NULL)
{
cout << "\n stack underflow" << endl;
exit(1);
}
else
{ x = top->data;
temp = top;
top = top->link;
temp->link = NULL;
free(temp);
return x;
}
}

3.

2. a) struct tree_node
{
char data;
tree_node* first_child;
tree_node* next_sibling;
} *tree_ptr;

b) void postorder(tree_ptr *p)
{
if (p==NULL) return;
if(p->child != NULL)
postorder(p->child);
cout<<p->data;
if(p->sibling != NULL)
Postorder(p->sibling);
}

c) B E F G C D A

3.

2. a) struct tree_node
{
char data;
tree_node* first_child;
tree_node* next_sibling;
} *tree_ptr;

b) void postorder(tree_ptr *p)
{
if (p==NULL) return;
if (p->child != NULL)
postorder(p->child);
cout<<p->data;
if (p->sibling != NULL)
Postorder (p->sibling);
}

c) B E F G C D A

```
treeNode {
        int data;
        treeNode *left;
        treeNode *right;
    }
```

1.

```
int CountInternal (treeNode * p)
{
  if (P == NULL || (p->left == NULL && P->right == NULL))
  return 0;
  return 1 + countInternal (P->left) + countInternal (P->right);

}
```

b) `bool remove_node(nodeptr & *head, int x)`

```
{
  if (head == NULL) return false;
  nodeptr* current = head;
  nodeptr* prev = head;
  while (current)
  {
    if (current->data != x)
    {
      prev = current;
      current = current->next;
    }
    prev->next = current->next;
      free(current);
    return true;

  }
  return false;
}
```

3) $\sum_{i=0}^{n/2} 1 = \frac{n}{2} - 0 + 1$

$= \frac{n}{2} + 1$

4) $\sum_{k=2}^{n} \sum_{j=1}^{n-1} 1$

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

$\sum_{k=2}^{n} (n-1)$

$\sum_{k=2}^{n} n - \sum_{k=2}^{n} 1$

$n[n-2+1] - [n-2+1]$

$n[n-1] - [n-1]$

$n^2 - n - n + 1$

$n^2 - 2n + 1$

$(n-1)^2$

5) $T(n) = (n-1) + T(n-1)$

$= 0$

$T(n) = T(n-1) + (n-1)$

$= T(n-2) + (n-2) + (n-1)$

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

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

$= T(n-k) + (n-k) + \cdots + (n-2) + (n-1)$

$n-k=1 \quad n=1+k$

$= T(1) + (1) + (n-(k-1)) + \cdots + (k-1) + (k)$

$= T(1) + 1 + 2 + 3 + \cdots + k$

$= 0 + \frac{k(k+1)}{2}$

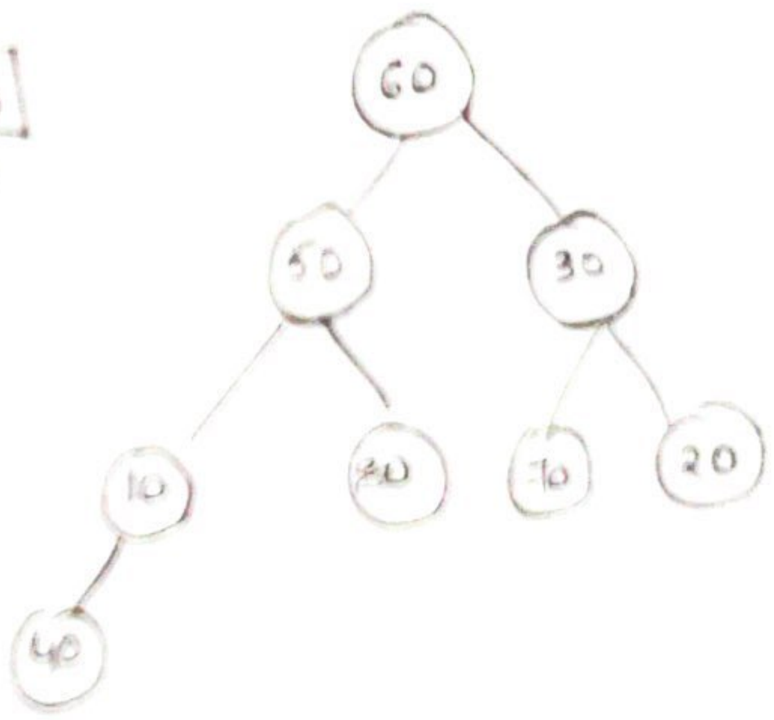
$= \frac{(n-1)(n)}{2}$

$= \frac{n^2 - n}{2}$

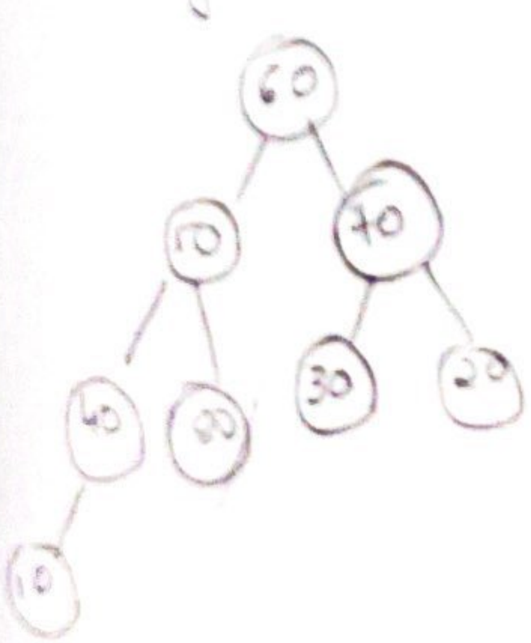
Spring 2015

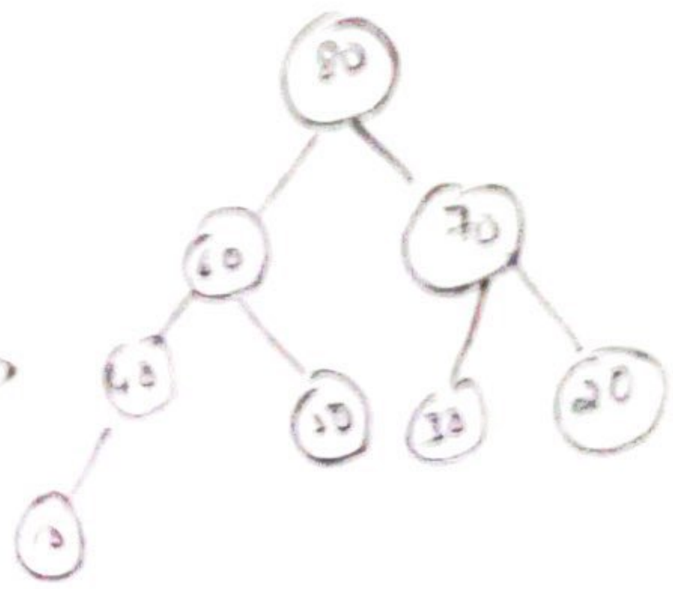
1.
```
int countnodes (struct node *root)
{
  if (root != NULL)
  {
    countnodes (root->left);
    count++;
    countnodes (root->right);
  }
  return count;
}
```

2. [60 | 50 | 30 | 10 | 40 | 30 | 20 | 40]



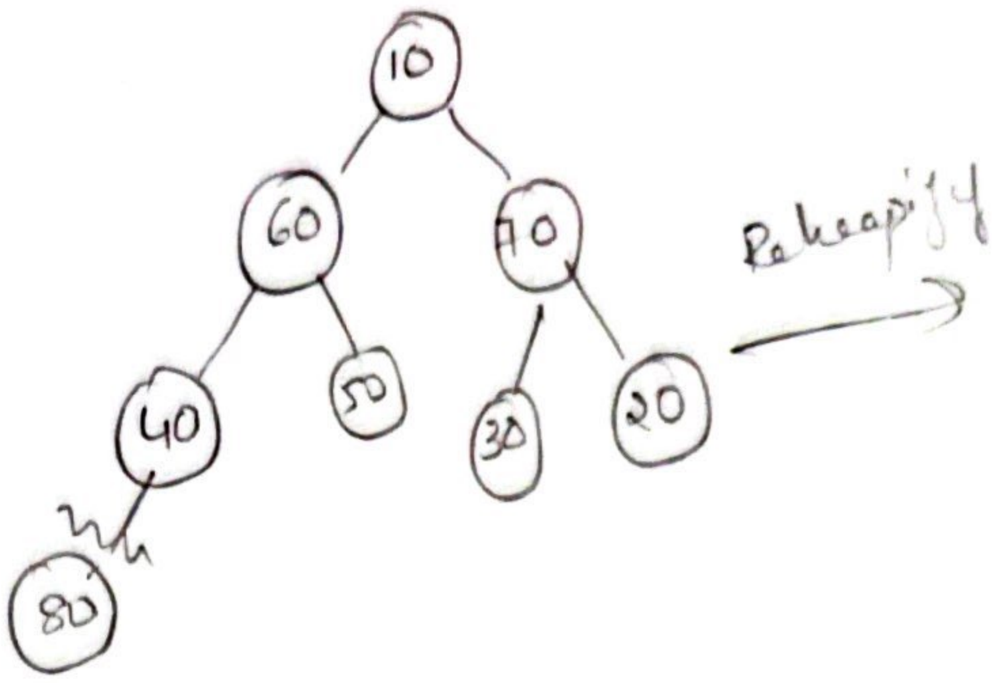
Transform above into max heap



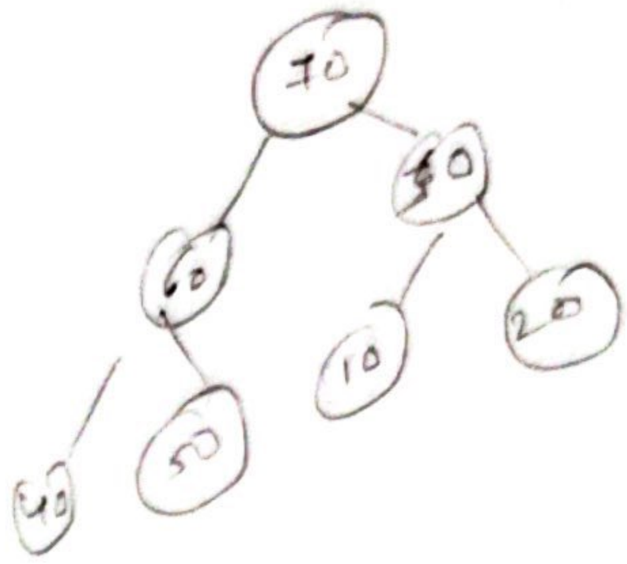


80 60 70 40 50 30 20

swap 80 and 10



Reheapify



70 60 30 40 50 10 20

10	60	70	40	50	30	20	80

Remove 70

swap 20 and 70

20 60 30 40 50 10 70 80

10 20 30 40 20 60 70 80

50 40 30 10 20 60 70 80

20 40 30 10 50 60 70 80

40 20 10 30 50 60 70 80

10 20 30 40 50 60 70 80

30 20 10 40 50 60 70 80

10 20 30 40 50 60 70 80

20 10 30 40 50 60 70 80

10 20 30 40 50 60 70 80

3 cy $\sum_{i=0}^{n/2} 1$

$= \frac{n}{2} - 0 + 1$

$= \frac{n}{2} + 1$

$= O(n)$

by $\sum_{k=2}^{n} \sum_{j=1}^{n-1} 1$

$\sum_{k=0}^{n} n - 1 - 1 + 1$

$\sum_{k=0}^{n} (n-1)$

$(n-1)(n-2+1)$

$(n-1)(n-1)$

by $\sum_{i=2}^{n-1} \left[\sum_{j=1}^{i} \right.$

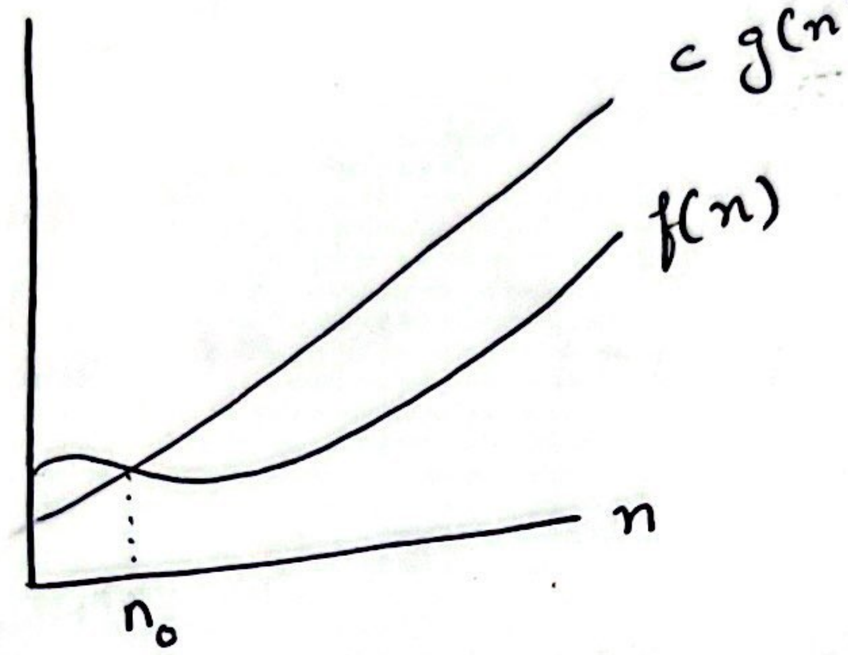
Fall 2014

1.a) Big-O Notation

The Big O notation defines an upper bound of an algorithm, it bounds a function only from above

$O(g(n)) = \{ f(n)$: there exists +ve constants c and n_0 such that $0 <= f(n) <= c * g(n) \quad \forall \quad n \geq n_0$

b)



$d \Rightarrow \log_b a$ - O(n)

b) $f(n) = n^2 - n$, $n \leq 23$

$g(n) = 2n - 1$

Sub $n = 20$ $\quad f(n) = (20)^2 - 20$

$= 20(20-1)$

Sub $n = 21$ $\quad f(n) = 21(20)$

$= 420$

$n^2 - n \leq c * (2n - 1)$

$23(22) \leq c * (23*2)$ $\quad c = 11$ ✓

$22(21) \leq c * (22*2) - 1$ $\quad c \geq 10.74$

$462 \leq c * 43$

$0 \leq f(n) \leq c * g(n) \quad \forall \quad n \geq n_0$

$n^2 - n \leq c * n$

$n - 1 \leq c$

Possibility c can be 1

$n^2 - n \leq 2n - 1$

val 20 $\quad 380 \leq 39$ X

val 21 $\quad 420 \leq 41$ X

val 22 $\quad 462 \leq 43$ X

val 23

$f(n) = 5n+7$ $\quad n > 23$ and n odd

Let $g(n) = n$

$n = 25$ $\quad 5(25)+7 \leq c * n$

$132 \leq c * 25$ ✓

$n = 21$ $\quad 5(21)+7 \leq c*21$

$105+7 \leq 13*21$

$112 \leq 273$ ✓

13
21
$\overline{13}$

26
$\overline{27}$

2 Q FIFO (Queue) using circular array a[0]...a[n-1]

```
Ans  class Queue
     {
        private:
              int [n];
              int front;
              int rear;
        public:
              Queue()
              {
                 int front = -1;
                 int rear = -1;
               }
};
```

```
Ans void insert (int x)
    {
      if(rear+1)%.n == = front
       return;
      else if (front = = rear == -1)
          front = rear = 0;
      else
         rear = (rear+1) %. n;
      a [rear] = x;
}
```

void dequeue()

{

if (front == rear == -1) return;

else if (front == rear)

front = rear = -1;

else

front = (front + 1) % n;

}

bool is_full()

{

return ((rear + 1) % n == front);

}

or better one

bool is_full()

{

if (front == (rear + 1) % n || front == 0 && rear == size - 1)

return true;

else return false;

}

3.

```
int Countkey(treeptr * p, int keyval)
{
    if(P==NULL) return 0;
    if(P->left==NULL && P->right==NULL && P->data==keyval)
    return 1;
    else
    return Countkey(P->left,keyval) + Countkey(P->right, keyval);
}
```

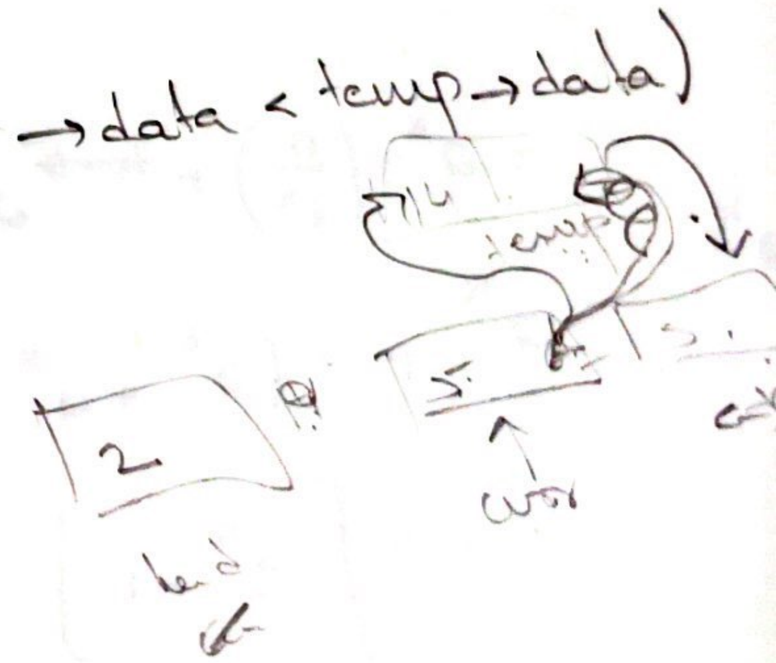
Spring 2014

1.
```
struct NodeType
{
  int data;
  struct Node* prev, *next;
};

void insert-double (*NodeType, int key)
{
  struct NodeType temp;
   temp → data = key;
   temp → prev = temp → next = NULL;
   struct NodeType* current;

if (head → data >= temp → data)
{
   temp → next = head;
   temp → next → prev = temp;
   head = temp;
  }
else
  {
   current = head;
   while (current → next != NULL && current → next → data < temp → data)
   current = current → next;

  temp → next = current → next;
  if (current → next != NULL)
      temp → next → prev = temp;
  current → next = temp;
  temp → prev = current;
}
}
```



2.

```
struct treeptr
{
   treeptr* left, right;

};

int countachildren(treeptr *p)
{
   if(P==NULL) return 0;
else
{
   int count=0;
   if(root→left!=NULL && root→right!=NULL)
     count++;
     return count + countachildren(P→left) + countachildren(P→rig
   }
}
```

3. $T(n) = 2T(n/2) + 5$

$= 2\left[2T\left(\frac{n}{2^2}\right) + 5\right] + 5$

$= 2^2 T\left(\frac{n}{2^2}\right) + 5 \cdot 2 + 5$

$\vdots$

$= 2^k T\left(\frac{n}{2^k}\right) + 2^{k-1} \cdot 5 + 2^{k-2} \cdot 5 + \cdots + 2^1 \cdot 5 + 2^0 \cdot 5$

$n = 2^k$

$= 2^k + 5\left(2^{k-1} + 2^{k-2} + \cdots + 2^1 + 2^0\right)$

$k = \log_2 n$

$= 2^k + 5\left[2^k - 1\right]$

$= n + 5\left[n - 1\right]$

$= n + 5n - 5$

$= 6n - 5$

Fall 2013

1. Node* insert (Node* root, int key)

```
{
   Node* newnode = newNode(key);
   Node* x = root;
   Node* y = NULL;
```

key should be *

```
while (x != NULL)
{
    y=x;
    if (key < x->key)
        x = x->left;
    else
        x = x->right;
  }
if (y == NULL)
    y = newnode;
elseif (key < y->key)
    y->left = newnode;
else
    y->right = newnode;
return y;
}
```

2. int partition(float a[], int first, int last)

```
{
    float x = a[first];
    int i = first - 1;
    int temp;
    for(int j = first; j <= last - 1; j++)
    {
      if(a[j] < x)
      {
        i++
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
      }
    }
    exchange a[i+1] with a[last]; //code
    return (i+1);
}
```

3. int median(int a, int b, int c)

```
{
  if(a > b)
  {
    if(b > c)
      return b;
    else if(a > c)
      return c;
    else
      return a;
  }
  else
  {
    if(a > c) return a;
    else if(b > c) return c;
    else return b;
  }
}
```

abc
acb
bac
bca
cab
cba

4 ; 3 comp

2 ; 2 comp

b) $avg = \frac{2 \cdot 2 + 4 \cdot 3}{6}$

$= \frac{16}{6}$

$= 8/3$

Spring 2013

1.
```
struct node
{
   int data;
   struct node* left, right;
};

int no_of_leaf(struct node* root)
{
    if(root == NULL)  return 0;
    if(node->right == NULL && node->left == NULL)
    return 1;
    else
    return  no-of-leaf(root->right) + no-of-leaf(root->left);
}
```

2.
```
void removeDuplicates (Node* head)
{
Node* current = head;
Node* next_next;
if(current == NULL) return;
while(current->next != NULL)
{
  if(current->data == current->next->data)
  {
     next_next = current->next->next;
     free(current->next);
     current->next = next_next;
  }
  else
     current = current->next;
  }
}
```

```
class Node
{
  Public:
      int data;
      Node* next;
};
```

2 3. // BFS adjacency list implementation

Fall 2012

a) string hashtable[ ];
int hashtablesize = N;

b) int find(string s)
{
int index = hash(s);

while(hashtable[index] != s && hashtable[index] != "")
{
index = (index + 1) % hashtablesize;
}
if(hashtable[index] == s)
return index;
else
return -1;
}

c) In worst case M collisions will occur
If the table size increases no. of collisions = M

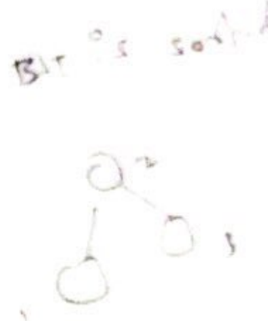
2.
```
struct node *newNode (int item)
{
    struct node *temp = (struct node *) malloc (sizeof (struct node
    temp → key = item;
    temp → left = temp → right = NULL;
    return temp;
}

struct node* insert(struct node* node, int key)
{
    if (node == NULL)  return newNode(key);

    if (key < node → key)
        node → left = insert (node → left, key);
    else if (key > node → key)
        node → right = insert (node → right, key);
    return node;
}

void insertBST (int a[], int n)
{
    struct node *root = NULL
    root = insert (root, a[0]);
    for (int i = 1 ; i < n ; i++)
        insert (root, a[i])
}
```

```
void inorder (struct node *root)
{
  if (root != NULL)
  {
    inorder (root → left);
    printf ("%d \n", root → key);
    inorder(root → right);
  }
}
```



3. $$\sum_{i=0}^{n-2} \left[\sum_{j=1}^{n} 1 + \sum_{k=i+2}^{n-1} 1 \right]$$

$$= \sum_{i=0}^{n-2} \left[n + (n-1) - (i+2) + 1 \right]$$

$$= \sum_{i=0}^{n-2} \left[n + n - 1 - i - 1 \right]$$

$$= \sum_{i=0}^{n-2} \left[2n - 2 - i \right]$$

$$= \sum_{i=0}^{n-2} 2n - 2 \sum_{i=0}^{n-2} - \sum_{i=0}^{n-2} i$$

$$= 2n\left[(n-2)+1 \right] - 2\left[n-1 \right] - \frac{(n-2)(n-1)}{2}$$

$$= 2n(n-1) - 2(n-1) - \frac{(n-2)(n-1)}{2}$$

$$= \frac{3}{2}n^2 - \frac{5}{2}n + 1$$

Spring 2012

Merge sort is best for LL but given in Q no new memory

Insertion sort is simple but

Quick sort is fast

10. //Sort linked list

```
struct node {
        int data;
        struct node *next;
    };

struct node *head, *tail = NULL;

void sortList()
{
  struct node *current = head, *index = NULL;
  int temp;

  if(head == NULL) return;

  else
  {
    while(current != NULL)
    {
      index = current → next;

      while(index != NULL)
      {
        if(current → data > index → data)
        {
          temp = current → data;

          current → data  = index → data;

          index → data = temp;

        }
      index = index → next;

      }
    current = current → next;
    }
  }
```

This is selection sort.

Spring 2012

1. //Sort linked list

```
struct node {
        int data;
        struct node *next;
    };

struct node *head, *tail = NULL;

void sortList()
{
  struct node *current = head, *index = NULL;
  int temp;
  if(head == NULL) return;
  else
  {
  while(current != NULL)
  {
    index = current → next;
    while(index != NULL)
    {
      if(current → data > index → data)
      {
        temp = current → data;
        current → data = index → data;
        index → data = temp;
      }
    index = index → next;
    }
  current = current → next;
  }
 }
}
```

Merge sort is best for LL but given in Q no new memory

Insertion sort is simple but

Quick sort is fast

This is selection sort.

```
class node
{
public :
        char data;
        node* left;
        node* right;

        node (int data)
        {
          this → data = data;
          this → left = NULL;
          this → right = NULL;
        }

};
void char_at_D (node *root, int D)
{
  if (root == NULL)
  return;
  if (D == 0)
  {
     cout << root → data << " ";
     return;
  }
  else
  {
     char_at_D (root → left, D-1);
     char_at_D (root → right, D-1);
  }
}
```

3. $T(n) = T(n/2) + \log(n)$

$T(1) = 1$

$$T(n) = T\left(\frac{n}{2^2}\right) + \log\left(\frac{n}{2}\right)$$

$$= T\left(\frac{n}{2^3}\right) + \log\left(\frac{n}{2^2}\right) + \log\left(\frac{n}{2}\right)$$

$$= T\left(\frac{n}{2^4}\right) + \log\left(\frac{n}{2^3}\right) + \log\left(\frac{n}{2^2}\right) + \log\left(\frac{n}{2}\right)$$

$$\vdots$$

$$= T\left(\frac{n}{2^k}\right) + \log\left(\frac{n}{2^{k-1}}\right) + \cdots + \log\left(\frac{n}{2^2}\right) + \log\left(\frac{n}{2}\right)$$

when $n = 2^k$ $\quad k = \log_2 n$

$$= T(1) + \log\left(2^{k-k+1}\right) + \cdots \log\left(2^{k-1}\right)$$

$$= 1 + \log(2)^1 + \log 2^2 + \cdots + \log\left(2^{k-1}\right)$$

$$= 1 + 1\log 2 + 2\log 2 + \cdots + (k-1)\log 2$$

$$= 1 + \log 2\,(1 + 2 + \cdots + k-1)$$

$$= 1 + \log 2\left(\frac{(k-1)(k)}{2}\right)$$

$$= 1 + \log 2\left(\frac{k^2 - k}{2}\right)$$

$$= 1 + \log 2\left(\frac{(\log_2 n)^2 - \log_2 n}{2}\right)$$

$$= 1 + \log 2\left($$

1. Binary search

```
int search (int a[], int l, int u, int x)
{
    if (u >= l)
    {
        int mid = l + (u-l)/2;
        if (a[mid] == x) return mid;
        if (a[mid] > x) return search (a, l, mid-1, x);
        return search (a, mid+1, u, x);
    }
    return -1;
}
```

2. int height (node* root)

```
{
    if (node == NULL) return 0;
    else
    {
        int ldepth = ~~root~~ height (node->left);
        int rdepth = height (node->right);
        if (ldepth > rdepth) return (ldepth+1);
        else return (rdepth+1);
    }
}
```

1. Binary search

```
int search (int a[], int l, int u, int x)
{
  if (u >= l)
  {
    int mid = l + (u-l)/2;
    if (a[mid] == x) return mid;
    if (a[mid] > x) return search (a, l, mid-1, x);
    return search (a, mid+1, u, x);
  }
  return -1;
}
```

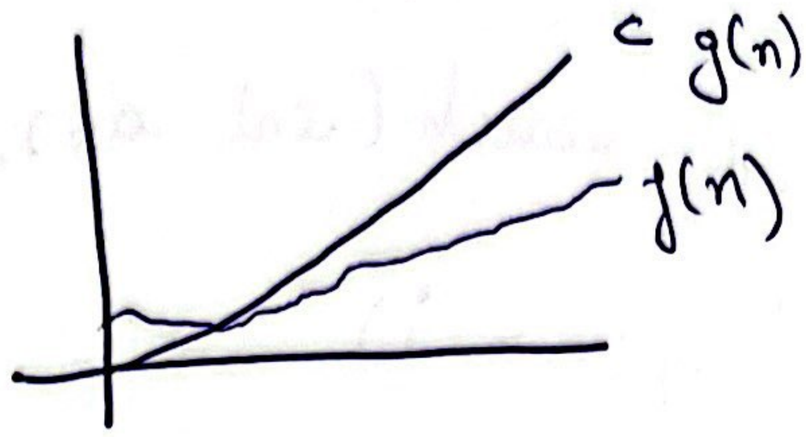
2. int height (node* root)

```
{
  if (node == NULL) return 0;
  else
  {
    int ldepth = ~~root~~ height (node->left);
    int rdepth = height (node->right);
    if (ldepth > rdepth) return (ldepth+1);
    else return (rdepth+1);
  }
}
```

3.

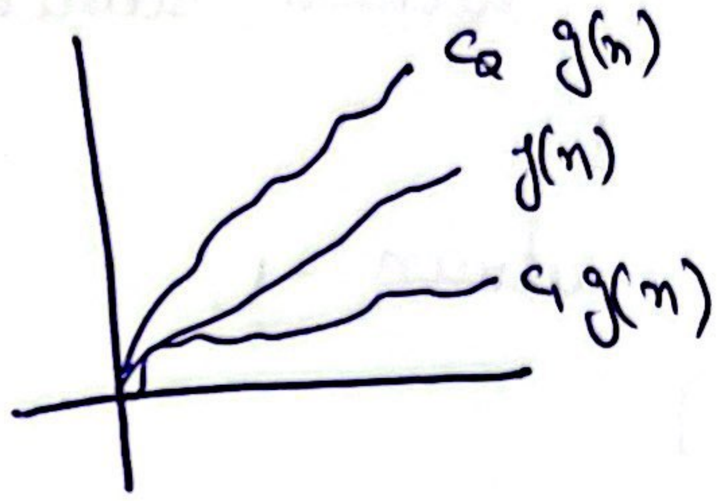
i, $f \in O(g)$

$f(n) = O(g(n))$ means that there are +ve constants c and k such that $0 \leq f(n) \leq c\,g(n)$ $\forall\ n \geq k$.



ii, $f \in \Theta(g)$

The theta notation bounds a functions from above and below, so it defines exact asymptotic behaviour.



b, $f(n) = 74n + 41$ $\quad g(n) = \frac{4}{5}n^2 - \frac{1}{8}n + 1$

i, $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{74n+41}{\frac{4}{5}n^2 - \frac{1}{8}n + 1}$

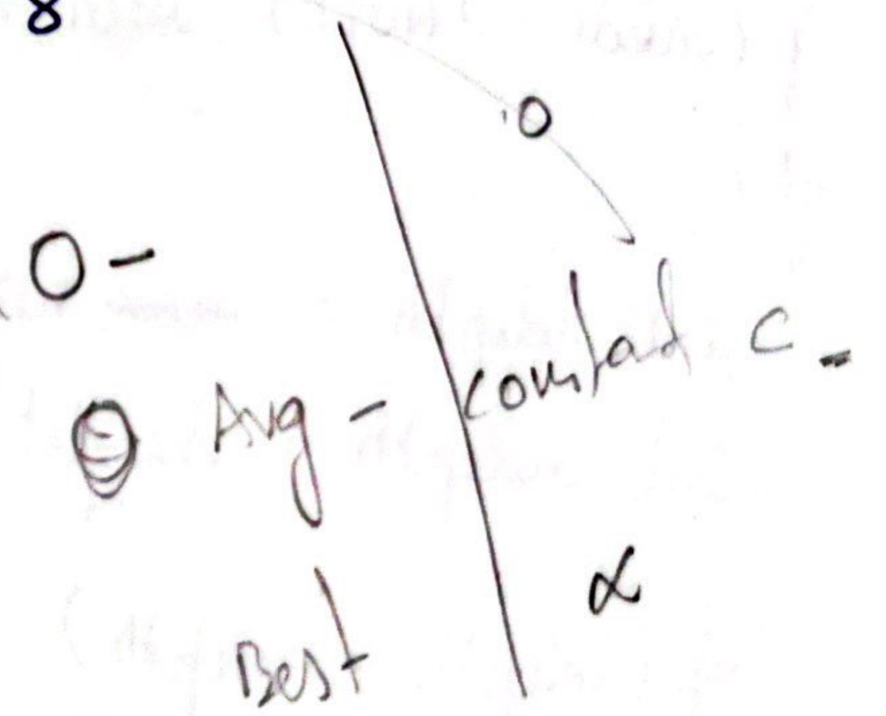
$= n\left[\frac{74 + \frac{41}{n}}{n}\right]$

$\lim_{n\to\infty} \frac{}{n\to\infty}\ n\left[\frac{\frac{4n}{5} - \frac{1}{8} + \frac{1}{n}}{n}\right]$

$\frac{74}{\infty + \frac{-1}{8} + 0}$

$74/\infty$

$\therefore c \in O(g)$



Picha method:

$f(n) = 74n + 41$ $\quad g(n) = \frac{4}{5}n^2 - \frac{1}{8}n + 1$

when $n=1$ $\quad 74 + 41 \leq c * \frac{4}{5} - \frac{1}{8} + 1$

$165 \leq c * 0.8 - 0.125 + 1$

$165 \leq c * 1.675$

$\therefore c \geq 98.56$

Big oh: $f(n) \leq g(n) * c$

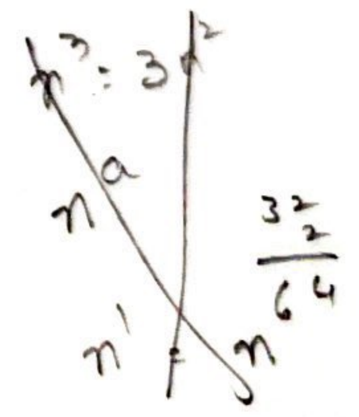
$74n + 41 \leq c * \frac{4}{5}n^2 - \frac{1}{8}n + 1$

$\frac{74n + 41}{\frac{4n^2}{5} - \frac{1}{8}n + 1} \leq c$

$\frac{40(74n + 41)}{32n^2 - 5n + 40} \leq c$

$\frac{74n + 41}{32n^2 - 5n + 40} \leq \frac{c}{40}$ $\quad C = \frac{c}{40}$

$\frac{74n + 41}{32n^2 - 5n + 40} \leq C$



let $n \to \alpha$ $\quad \frac{\alpha}{\alpha}$ L'Hospital

$\frac{74 + 41}{64n - 5 + 40} \leq C$

$$\frac{115}{6n+35} \leq C$$

when $n \to \infty$

$$\frac{115}{\infty}$$

$$0 \leq \frac{c}{40}$$

$\therefore$ Big oh True.

by Ω $f(n) \geq c * g(n)$

$$\frac{115}{\infty} \geq \frac{c}{40}$$

$$0 \geq \frac{c}{40}$$

$c = 1$ X

$\therefore$ False.