1. log n, (log n)^2, n^3, (3/2)^n, 4^n, n!
2. n

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 3 |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 8 |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 |  |  |  |  |  |



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 4 |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 4 | 1 |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 4 | 1 | 3 |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 4 | 1 | 3 |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 1 | 3 | 8 |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 1 | 3 | 8 |  |  |

1. Hh

//insertion at beginning

struct node \*newNode;

newNode = malloc(sizeof(struct node));

newNode->data = sizeoflist;

newNode->next = head;

head = newNode;

//insertion at end

struct node \*newNode;

newNode = malloc(sizeof(struct node));

newNode->data = sizeofList;

newNode->next = NULL;

struct node \*temp = head;

while(temp->next != NULL){

temp = temp->next;

}

temp->next = newNode;

//Insert

struct node \*newNode;

newNode = malloc(sizeof(struct node));

newNode->data = sizeofList;

struct node \*temp = head;

for(int i=2; i < position; i++) {

if(temp->next != NULL) {

temp = temp->next;

}

}

newNode->next = temp->next;

temp->next = newNode;

//DELETION

//deletion from head

head = head→next;

//deletion from the end

struct node\* temp = head;

while(temp->next->next!=NULL){

temp = temp->next;

}

//set the last node to null since we deleted the contents

temp->next = NULL

1. **Algorithm Tree-Predecessor (root)  
   Start**

**if root.left != null // if root left not equal to null then return treemax  
         return TreeMax(root.left)  
     
     Node p = root.parent; ` // node p is root parent   
     while p != null && root == p.left // checks if p has a value and is equal to left parent   
         root = p;  
         p = p.parent;  
     
     return p  
  
End**