**Homework lecture**

**Complexity analyses**

**1)**

Ascending order: 2^10 -> 3n+100logn -> 4n -> nlogn -> 4nlogn+2n -> n^2+10n -> n^3 -> 2^logn >2^n.

**2)**

Procedure exponential(int n){

Long long temp=1;

For (int i=1 to i= n-1)

temp\*=2;

return temp;

}

Complexity: O(n);

Procedure exponential(int n){

Long long temp =1;

Int i=1;

If (n%2==1)

Temp\*=2;

n--;

While (n%2==0){

Temp\*=temp;

If (n%2==1)

Temp\*=2;

n--;

}

Return temp;

}

Complexity: O(logn)

**3)**

Int Arr[1000];

Int count=0;

Procedure push(int arr, int value){

If (count<1000)

Arr[count]=value;

Count++;

Else print(“full”);

}

Complexity: O(1);

Procedure top(int arr){

If (count >0)

Return arr[0];

Else print(“none”);

}

Complexity: O(1)

Procedure pop(int arr){

If (count ==0)

Print (“ empty”)

Else for (int i=0 to count)

Arr[i]=arr[i+1];

}

Complexity: O(n)

**4)**

Node\* a{

Int value;

Node\* next;

}

singleLink\* l{

node\*head;

node\* tail;

}

Procedure push(singleLink\* l,value){

if (l->head==null)

l->head=new node

l->head->value=value

l->head->next=null;

l->tail=l->head;

else

node\* temp=new node;

l->tail->next=temp;

temp->value=value;

temp->next=null;

l->tail=temp;

return l->head;

}

Complexity: O(1);

Procedure top(singleLinked\* l){

If (l->head=null)

Do nothing

Return l->tail->value;

}

Complexity: O(1)

Procedure pop(singleLinked\*l){

If (l->head=null)

Do nothing

Node\* temp=l->head

While (temp->next!=l->tail)

Temp=temp->next;

l->tail=temp;

l->tail->next=null;

return l;

}

Complexity: O(n);

**5)**

Function push(element):

If top is equal to max\_size - 1:

Print "Stack Overflow: Cannot push element"

Else:

Increment top by 1

stack[top] = element

O(1)

Function pop():

If top is equal to -1:

Print "Stack Underflow: Cannot pop element"

Return null // or a suitable error value

Else:

element = stack[top]

Decrement top by 1

Return element

O(1)

Function top():

If top is equal to -1:

Print "Stack is empty: Cannot retrieve top element"

Return null // or a suitable error value

Else:

Return stack[top]

O(1)

**6)**

Node\* a{

Int value;

Node\* next;

}

Singlink\* l{

Node\* head;

Node\* tail;}

Push(Singlink\* l, int value){

If (l->head=null)

l->head=new node;

l->head->value=value;

l->head->next=null;

l->tail=l->head;

else

node\* temp=new node;

temp->value=value;

temp->next=null

l->tail->next=temp;

l->tail=temp;

return l;

}

Complexity: O(1);

Top(Singlink\* l){

If (l->head==null)

Do nothing

Else return l->head->value;

}

Complexity: O(1)

Pop(singling\* l){

Node\* temp=l->head;

l->head=l->head->next;

free(temp)

}

Complexty: O(1)