Iterative and Recursive Methods

1. Write a C/C++ program to print Fibonacci series upto nth term using iteration also compute time complexity.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
   int cost=0;
   int a,b,c,n;
   printf("Enter the range of series\n");
   scanf("%d",&n);
   a=0;
   b=1;
   printf("%d\t",a);
   printf("%d\t",b);
   cost+=4;
   for(int i=3;i<=n;i++){
      c=a+b;
      printf("%d\t",c);
      a=b;
      b=c;
      cost+=4;
   }
   printf("\n");
   printf("Total cost=%d\n",cost);
   return 0;
}
```

OUTPUT:

```
Enter the range of series
6
0       1       1       2       3       5
Total cost=20
```

---

2. Write a C/C++ program to print Fibonacci series upto nth term using recursion also compute the time complexity in terms of input size.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int count=0;
void fibo(int n,int a,int b){
   if(n==0 || n==1){
      count++;
      return;
   }
   printf("%d\t",a+b);
   count+=2;
   fibo(n-1,b,a+b);
}
void print_fibo(int n){
   printf("%d\t%d\t",0,1);
   count+=1;
   fibo(n-1,0,1);
   printf("\n");
}
int main(){
   printf("Enter the number of terms of the series you want to print\n");
   int n;
   scanf("%d",&n);
   print_fibo(n);
   printf("Cost for input size %d = %d\n",n,count);
   return 0;
}
```

OUTPUT:

Enter the number of terms of the series you want to print

5

0 1 1 2 3

Cost for input size 5 = 8

Searching

3. Write a C/C++ program to search an element using linear search in an array also compute time complexity for an input of size N.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int cost=0;
int linsearch(int *arr,int size,int item){
    if(arr[size]==item && size>=0){
        cost+=2;
        return size;
    }
    else if(size<0){
        cost++;
        return 0;
    }
    else{
        return linsearch(arr,size-1,item);
    }
}
int main(){
    int *arr;
    int n;
    printf("Enter the size of the array\n");
    cost++;
    scanf("%d",&n);
    arr=(int *)malloc(sizeof(int)*n);
    printf("Enter array elements\n");
    cost+=3;
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
        cost++;
    }
    printf("Enter the item you want to search for\n");
    int item;
    scanf("%d",&item);
    int found=linsearch(arr,n-1,item);
    cost+=3;
    if(found==0){
        printf("item not found:)\n");
        cost++;
```

```
    }
    else{
        printf("Item found at index:%d\n",found);
    }
    printf("Cost=%d\n",cost);
    return 0;
}
```

OUTPUT:

Enter the size of the array

8

Enter array elements

14 22 126 9 281 78 39 101

Enter the item you want to search for

78

Item found at index:5

Cost=17

---

4. Recursive Write a C/C++ program to perform binary search onan array of size N and compute time complexity for size N.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int cost=0;
int binary_search(int *arr,int lb,int ub,int item){
   int mid=(lb+ub)/2;
   cost++;
   if(lb>ub){
      cost++;
      return 0;
   }
   else if(item>arr[mid]){
      cost++;
      return binary_search(arr,mid+1,ub,item);
   }
   else if(item<arr[mid]){
      cost++;
      return binary_search(arr,lb,mid-1,item);
   }
   else{
      return mid;
   }

}
int main(){
   int *arr;
   int n;
   printf("Enter the size of the array\n");

   scanf("%d",&n);
   arr=(int *)malloc(sizeof(int)*(n+1));
   printf("Enter array elements; it must be sorted\n");

   for(int i=1;i<=n;i++){
      scanf("%d",&arr[i]);

   }
   printf("Enter the item you want to search for\n");
   int item;
   scanf("%d",&item);
   int found=binary_search(arr,1,n,item);
```

```
    if(found==0){
        printf("item not found:)\n");

    }
    else{
        printf("Item found at position:%d\n",found);
    }
    printf("Cost=%d\n",cost);
    return 0;
}
```

OUTPUT:

Enter the size of the array

6

Enter array elements; it must be sorted

14 32 49 71 86 108

Enter the item you want to search for

49

Item found at position:3

Cost=1

---

5. Write a C/C++ program to perform **bubble sort** on an integer array to sort it in ascending order and compute the time complexity for an input of size N.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int cost=0;
void Bubble_sort(int *arr,int n){
    int i,j;
    for(i=0;i<=n-1;i++){
        for(j=0;j<n-1-i;j++){
            if(arr[j]>arr[j+1]){
                int temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
            }
            cost+=4;
        }
    }
}
int main(){
    int *arr;
    int n;
    printf("Enter the size of the array\n");

    scanf("%d",&n);
    arr=(int *)malloc(sizeof(int)*(n+1));
    printf("Enter array elements\n");

    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);

    }
    printf("Entered array-------\n");
    for(int i=0;i<n;i++){
        printf("%d\t",arr[i]);
    }
    printf("\n");
    Bubble_sort(arr,n);
    printf("sorted array-------\n");
    for(int i=0;i<n;i++){
```

```
        printf("%d\t",arr[i]);
    }
    printf("\n");
    printf("Cost=%d\n",cost);
    return 0;
}
```

OUTPUT:

```
Enter the size of the array
10
Enter array elements
45 23 90 8 143 52 19 71 33 65
Entered array-------
45      23      90      8       143     52      19      71      33      65
sorted array-------
8       19      23      33      45      52      65      71      90      143
Cost=180
```

---

6. Write a C/C++ program to perform **insertion sort** on an integer array to sort it in ascending order and compute the time complexity for an input of size N.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int s;
    printf("Enter the size of array\n");
    scanf("%d",&s);
    int i,j,key;
    int *arr=(int *)malloc(sizeof(int)*s);
    printf("Enter array elements\n");
    for(int i=0;i<s;i++){
        scanf("%d",&arr[i]);
    }
    for(i=0;i<s;i++){
        printf("%d\t",arr[i]);
    }
    printf("\n");
    for(i=1;i<s;i++){
        key=arr[i];
        for(j=i-1;j>=0;j--){
            if(key< arr[j])
                arr[j+1]=arr[j];
            else
                break;
```

```
        }
        arr[j+1]=key;
    }
     printf("Sorted array\n");
     for(i=0;i<s;i++)
        printf("%d\t",arr[i]);
    printf("\n");
    return 0;
}
```

Output:

```
Enter the size of array
7
Enter array elements
12 31 6 90 505 22 80
12      31      6       90      505     22      80
Sorted array
6       12      22      31      80      90      505
```

---

7. Write a C/C++ program to perform **selection sort** on an integer array to sort it in ascending order and compute the time complexity for an input of size N.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int cost=0;
    int *arr;
    int n,i,j,min,mini;
    printf("Enter the size of the array\n");
    scanf("%d",&n);
    arr=(int *)malloc(sizeof(int)*n);
    printf("Enter the array element\n");
    cost+=5;
    for(i=0;i<n;i++){
        scanf("%d",&arr[i]);
        cost+=3;
    }
    printf("Entered array-------\n");
    for(int i=0;i<n;i++){
        printf("%d\t",arr[i]);
        cost+=3;
    }
    printf("\n");
```

```
    for(i=0;i<n;i++){
        min=arr[i];
        mini=i;
        cost+=5;
        for(j=i+1;j<n;j++){
            cost+=2;
            if(arr[j]<min){
                min=arr[j];
                mini=j;
                cost+=3;
            }
        }
        if(arr[i]!=min){
            int t=arr[i];
            arr[i]=arr[mini];
            arr[mini]=t;
            cost+=4;
        }
    }
    printf("sorted array-------\n");
    for(int i=0;i<n;i++)
        printf("%d\t",arr[i]);
    printf("\n");
    printf("Total cost=%d\n",cost);
    return 0;
}
```

OUTPUT:

```
Enter the size of the array
8
Enter the array element
20 48 17 34 102 1 444 78
Entered array-------
20      48      17      34      102     1       444     78
sorted array-------
1       17      20      34      48      78      102     444
Total cost=197
```

8. Write a C/C++ program to perform **merge sort** on an integer array to sort it in ascending order and compute the time complexity for an input of size N.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int cost=0;
void Merge(int *arr,int l,int mid,int h){
   int l1=mid-l+1;
   int l2=h-mid;
   int *arr1=(int *)malloc(sizeof(int)*l1);
   int *arr2=(int *)malloc(sizeof(int)*l2);
   int i;
   for(i=l;i<=mid;i++){
      arr1[i-l]=arr[i];
      cost+=3;
   }
   for(i=mid+1;i<=h;i++){
      arr2[i-(mid+1)]=arr[i];
      cost+=3;
   }
   i=0;
   int j=0;
   int k=l;
   while(i<l1 && j<l2){
      cost+=2;
      if(arr1[i]<arr2[j]){
         cost+=2;
         arr[k++]=arr1[i];
         i++;
      }
      else{
         arr[k++]=arr2[j];
         j++;
         cost+=2;
      }
   }
   while(i<l1){
      arr[k++]=arr1[i];
      i++;
      cost+=2;
   }
   while(j<l2){
      arr[k++]=arr2[j];
      j++;
```

```
        cost+=2;
    }
}
void Mergesort(int * arr,int l,int h){
    int mid;
    if(l<h){
        cost+=2;
        mid=(l+h)/2;
        Mergesort(arr,l,mid);
        Mergesort(arr,mid+1,h);
        Merge(arr,l,mid,h);
    }
}
int main(){
    int *arr;
    int n,i;
    printf("Enter the size of the array\n");
    scanf("%d",&n);
    arr=(int *)malloc(sizeof(int)*n);
    printf("Enter the array element\n");
    for(i=0;i<n;i++)
         scanf("%d",&arr[i]);
    printf("Entered array-------\n");
    for(int i=0;i<n;i++)
         printf("%d\t",arr[i]);
    printf("\n");
    Mergesort(arr,0,n-1);
     printf("Sorted array-------\n");
    for(int i=0;i<n;i++)
        printf("%d\t",arr[i]);
    printf("\n");
    printf("Total cost=%d\n",cost);
    return 0;
}
```

OUTPUT:

```
Enter the size of the array
8
Enter the array element
20 48 17 34 102 1 444 78
Entered array-------
20      48      17      34      102     1       444     78
Sorted array-------
1       17      20      34      48      78      102     444
Total cost=164
```

9. Write a C/C++ program to perform **quick sort** on an integer array to sort it in ascending order and compute the time complexity for an input of size N.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int cost=0;
void swap(int *x,int *y){
   int t=*x;
   *x=*y;
   *y=t;
   cost+=3;
}
int partition(int *arr,int l,int h){
   int pivot=arr[h];
   int i=l-1;
   int j;
   for(j=l;j<=h;j++){
      cost+=2;
      if(arr[j]<pivot){
         i++;
         swap(&arr[i],&arr[j]);
         cost+=3;
      }
   }
   i++;
   swap(&arr[i],&arr[h]);
   return i;
}
void Quicksort(int *arr,int l,int h){
   int i;
   if(l<h){
      cost+=1;
      i=partition(arr,l,h);
      Quicksort(arr,l,i-1);
      Quicksort(arr,i,h);
   }
}
int main(){
   int *arr;
   int n,i;
   printf("Enter the size of the array\n");
   scanf("%d",&n);
   arr=(int *)malloc(sizeof(int)*n);
   printf("Enter the array element\n");

   for(i=0;i<n;i++)
```

```
        scanf("%d",&arr[i]);
    printf("Entered array-------\n");
    for(int i=0;i<n;i++)
      printf("%d\t",arr[i]);
    printf("\n");
    Quicksort(arr,0,n-1);
     printf("Sorted array-------\n");
    for(int i=0;i<n;i++)
      printf("%d\t",arr[i]);
    printf("\n");
    printf("Total cost=%d\n",cost);
    return 0;
  }
```

OUTPUT:

```
Enter the size of the array
7
Enter the array element
34 12 183 44 90 1800 1
Entered array-------
34      12      183     44      90      1800    1
Sorted array-------
112     34      44      90      183     1800
Total cost=144
```

---

10. Write a C/C++ program to perform **count sort** on an integer array to sort it in ascending order and compute the time complexity for an input of size N.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int cost=0;
int getMax(int *arr,int n){
   int i,max;
   max=arr[0];
   for(i=0;i<n;i++){
      cost+=2;
      if(arr[i]>max){
         cost+=2;
         max=arr[i];
      }
   }
   return max;
}
void count_sort(int *arr,int n){
   int max=getMax(arr,n);
   int *count=(int *)calloc(sizeof(int),(max+1));
```

```
    int *brr=(int *)calloc(sizeof(int),n+1);
    int i;
    for(i=0;i<n;i++){
        cost+=3;
        count[arr[i]]++;
    }
    for(i=1;i<=max;i++){
        cost+=3;
        count[i]=count[i-1]+count[i];
    }
    for(i=0;i<n;i++){
        cost+=4;
        brr[count[arr[i]]]=arr[i];
        count[arr[i]]--;
    }
    int k=0;
    for(i=1;i<=n;i++){
        cost+=3;
        arr[k++]=brr[i];
    }
}
int main(){
    int *arr;
    int n,i;
    printf("Enter the size of the array\n");
    scanf("%d",&n);
    arr=(int *)malloc(sizeof(int)*n);
    printf("Enter the array element\n");

    for(i=0;i<n;i++)
      scanf("%d",&arr[i]);
    printf("Entered array-------\n");
    for(int i=0;i<n;i++)
        printf("%d\t",arr[i]);
    printf("\n");
    count_sort(arr,n);
     printf("Sorted array-------\n");
    for(int i=0;i<n;i++)
        printf("%d\t",arr[i]);
    printf("\n");
    printf("Total cost=%d\n",cost);
    return 0;
}
```

OUTPUT:

```
Enter the size of the array
9
Enter the array element
26 5 18 51 17 42 59 64 72
Entered array-------
26      5       18      51      17      42      59      64      72
Sorted array-------
5       17      18      26      42      51      59      64      72
Total cost=332
```

---

11. Write a C/C++ program to perform **radix sort** on an integer array to sort it in ascending order and compute the time complexity for an input of size N.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int cost=0;
int getMax(int *arr,int n){
   int i,max;
   max=arr[0];
   for(i=0;i<n;i++)
   {
      cost+=2;
      if(arr[i]>max)
      {
         cost+=2;
         max=arr[i];
      }
   }
   return max;
}
void Countsort(int *arr,int m,int pos)
{
   int *count=(int *)calloc(10,sizeof(int));
   int *brr=(int *)malloc(sizeof(int)*(m+1));
   int i;
   for(i=0;i<m;i++)
   {
      count[(arr[i]/pos)%10]++;
      cost+=3;
   }
   for(i=1;i<10;i++)
   {
```

```
        count[i]=count[i-1]+count[i];
        cost+=3;
    }
    for(i=m-1;i>=0;i--)
    {
        brr[count[(arr[i]/pos)%10]]=arr[i];
        count[(arr[i]/pos)%10]--;
        cost+=4;

    }
    int k=0;
    for(i=1;i<=m;i++)
    {
        arr[k++]=brr[i];
        cost+=3;
    }
}
void Radixsort(int *arr,int m)
{
    int max=getMax(arr,m);
    int pos;
    for(pos=1;max/pos>0;pos=pos*10)
    {
        Countsort(arr,m,pos);
    }
}
int main()
{
     int *arr;
    int n,i;
    printf("Enter the size of the array\n");
    scanf("%d",&n);
    arr=(int *)malloc(sizeof(int)*n);
    printf("Enter the array element\n");

    for(i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);

    }
    printf("Entered array-------\n");
    for(int i=0;i<n;i++)
    {
        printf("%d\t",arr[i]);
```

```
    }
    printf("\n");
    Radixsort(arr,n);
     printf("Sorted array-------\n");
    for(int i=0;i<n;i++)
    {
       printf("%d\t",arr[i]);

    }
    printf("\n");
    printf("Total cost=%d\n",cost);
    return 0;
}
```

OUTPUT:

```
Enter the size of the array
7
Enter the array element
85 29 182 148 191 172 70
Entered array-------
85      29      182     148     191     172     70
Sorted array-------
29      70      85      148     172     182     191
Total cost=309
```

---

12. Write a C/C++ program to **insert an element into heap**, also compute time complexity for an input of size N.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int cost=0;
void swap(int *x,int *y)
{
   int t=*x;
   *x=*y;
   *y=t;
   cost+=3;
}
int* insertintoheap(int *arr,int n,int x)
{
   n=n+1;
   int *brr=(int *)calloc(n+1,sizeof(int));
   brr[0]=0;
   for(int i=1;i<n;i++)
   {
      brr[i]=arr[i];
      cost+=3;
   }
   brr[n]=x;
   int  i=n;
   int parent;
   while(i>1)
   {
      parent=i/2;
      if(brr[parent]<brr[i])
      {
         swap(brr+parent,brr+i);
         i=parent;
         cost+=2;
      }
      else{
         cost++;
         break;
      }
   }
   return brr;
}
int main()
```

```
{
   int n=0;
   char ch;
   int *arr;
   do{
      int value;
      printf("Enter the value you want to insert\n");
      scanf("%d",&value);
      arr=insertintoheap(arr,n,value);
      n++;
      printf("Do you want to insert again y/n\n");
      scanf("%c",&ch);
      scanf("%c",&ch);
   }while (ch=='y');
    for(int i=1;i<=n;i++)
   {
      printf("%d\t",arr[i]);
   }
   printf("\n");
   printf("Total cost=%d",cost);
   return 0;
}
```

Output:

```
Enter the value you want to insert
47
Do you want to insert again y/n
y
Enter the value you want to insert
183
Do you want to insert again y/n
y
Enter the value you want to insert
51
Do you want to insert again y/n
y
Enter the value you want to insert
102
Do you want to insert again y/n
y
Enter the value you want to insert
61
Do you want to insert again y/n
y
Enter the value you want to insert
94
```

```
Do you want to insert again y/n
n
183     102     94      47      61      51
Total cost=64
```

---

13. Write a C/C++ program to **delete the N element** from a heap, also compute time complexity for those N elements.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int cost=0;
void swap(int *x,int *y)
{
   int t=*x;
   *x=*y;
   *y=t;
   cost+=3;
}
int* insertintoheap(int *arr,int n,int x)
{
   n=n+1;
   int *brr=(int *)calloc(n+1,sizeof(int));
   brr[0]=0;
   for(int i=1;i<n;i++)
   {
      brr[i]=arr[i];
      cost+=3;
   }
   brr[n]=x;
   int  i=n;
   int parent;
   while(i>1)
   {
      parent=i/2;
      if(brr[parent]<brr[i])
      {
         swap(brr+parent,brr+i);
         i=parent;
         cost+=2;
      }
      else{
         cost++;
         break;
      }
   }
```

```
    return brr;
}
void delete_top(int *arr,int n)
{
    swap(&arr[1],&arr[n]);
    n=n-1;
    int i,l,r;
    i=1;
    while(i<n)
    {
        l=2*i;
        r=2*i+1;
        if(arr[l]>arr[r] && l<=n){
            swap(&arr[l],&arr[i]);
            i=l;
            cost+=3;
        }
        else if(arr[r]>arr[l] && r<=n)
        {
            swap(&arr[r],&arr[i]);
            cost+=2;
            i=r;
        }
        else{
            cost+=1;
            return;
        }
    }
}
int main()
{
    int n=0;
    char ch;
    int *arr;
    do{
        int value;
        printf("Enter the value you want to insert\n");
        scanf("%d",&value);
        arr=insertintoheap(arr,n,value);
        n++;
        printf("Do you want to insert again y/n\n");
        scanf("%c",&ch);
        scanf("%c",&ch);
    }while (ch=='y');
     for(int i=1;i<=n;i++)
```

```
    {
        printf("%d\t",arr[i]);
    }
    printf("\n");

    while(n>=1)
    {
        delete_top(arr,n);
        n--;
            if(n>0){
        printf("The heap after deleting----\n");
        for(int i=1;i<=n;i++)
        {
            printf("%d\t",arr[i]);
        }
        printf("\n");
        }
        else{
            printf("Heap is empty....:(\n");

        }
    }
    printf("Total cost=%d\n",cost);
    return 0;
}
```

Output:

```
Enter the value you want to insert
47
Do you want to insert again y/n
y
Enter the value you want to insert
183
Do you want to insert again y/n
y
Enter the value you want to insert
51
Do you want to insert again y/n
y
Enter the value you want to insert
102
Do you want to insert again y/n
y
Enter the value you want to insert
61
```

```
Do you want to insert again y/n
y
Enter the value you want to insert
94
Do you want to insert again y/n
n
183     102     94      47      61      51
The heap after deleting----
102     61      94      47      51
The heap after deleting----
94      61      51      47
The heap after deleting----
61 47   51
The heap after deleting----
51 47
The heap after deleting----
47
Heap is empty....:(
Total cost=107
```

---

14. Write a C/C++ program to **build a heap using heapify** and use it to perform heap sort, also compute the time complexity for an input of size N.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int cost=0;
void swap(int *x,int *y)
{
   int t=*x;
   *x=*y;
   *y=t;
   cost+=3;
}
void heapify(int *arr,int n,int i)
{
   int largest=i;
   int l=2*i;
   int r=(2*i)+1;
   cost+=3;
   if(l<=n && arr[l]>arr[largest])
   {
      largest=l;
      cost+=3;
   }
```

```
    else if(r<=n && arr[r]>arr[largest])
    {
        largest=r;
        cost+=3;
    }
    cost+=1;
    if(largest!=i)
    {

        swap(&arr[i],&arr[largest]);
        heapify(arr,n,largest);
    }
}
void BuildHeap(int *arr,int n)
{
    int i;
    cost++;
    for(i=n/2;i>=1;i--)
    {
        cost+=2;
        heapify(arr,n,i);
    }
}
void delete_sort(int *arr,int n)
{
    int i;
    cost++;
    for(i=n;i>=1;i--)
    {
        cost+=2;
        swap(&arr[1],&arr[i]);
        heapify(arr,i-1,1);
    }
}
int main()
{
    printf("Enter the size of array\n");
    int n;
    scanf("%d",&n);
    int *arr=(int*)malloc(sizeof(int)*(n+1));
    printf("Enter the array elements\n");

    for(int i=1;i<=n;i++)
    {
        scanf("%d",&arr[i]);
```

```
    }
     for(int i=1;i<=n;i++)
    {
        printf("%d\t",arr[i]);
    }
    printf("\n");
    printf("Heapified version-----\n");
    BuildHeap(arr,n);
     for(int i=1;i<=n;i++)
    {
        printf("%d\t",arr[i]);
    }
     printf("\n");
    BuildHeap(arr,n);
    delete_sort(arr,n);
    printf("Sorted version\n");
    for(int i=1;i<=n;i++)
    {
        printf("%d\t",arr[i]);
    }
    printf("\n");
    printf("Total cost=%d\n",cost);
    return 0;
}
```

OUTPUT:

```
Enter the size of array
8
Enter the array elements
20 90 166 28 46 131 182 137
20      90      166     28      46      131     182     137
Heapified version-----
137     90      182     28      46      131     166     20
Sorted version
20      46      28      131     137     166     90      182
Total cost=263
```

---

15. Write a C/C++ program to implement dynamic array. First take maximum length of array from user input. Then start by creating array of size 1, and start taking input. Every time the array is full, double its capacity. Use amortize analysis (aggregate) to calculatetime complexity of the program.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int size,n,cost=0;
int *create()
{
   int size1;
   if(size==0)
   {
      size1=1;
   }
   else{
      size1=2*size;
   }
   int *ar=(int *)calloc(size1,sizeof(int));
   cost+=1;
   size=size1;
   return ar;
}
void copy(int *arr,int *ar)
{
   int i;
   for(i=0;i<n;i++)
   {
      ar[i]=arr[i];
   }
   cost++;

}
int *insert(int *arr,int item)
{
   int *ar;
   if(n==size)
   {
      ar=create();
      copy(arr,ar);
      arr=ar;
      arr[n++]=item;
```

```
    }
    else if(n<size){
        arr[n++]=item;
    }
    cost++;
    return arr;
}
int main()
{
    int* arr;
    size=0;n=0;
    char ch;
    int item;
    do{
        printf("Enter the item you want to insert\n");
        scanf("%d",&item);
        arr=insert(arr,item);
        for(int i=0;i<n;i++)
        {
            printf("%d\t",arr[i]);

        }
        printf("\n");
        printf("Do you want to insert again y/n\n");
        scanf("%c",&ch);
        scanf("%c",&ch);
    }while(ch=='y');
    printf("%d   %d\n",n,cost);
    int averagecost=cost/n;
    printf("%d\n",averagecost);
    return 0;

}
```

OUTPUT:

```
Enter the item you want to insert
81
81
Do you want to insert again y/n
y
Enter the item you want to insert
36
81 36
Do you want to insert again y/n
y
Enter the item you want to insert
```

```
10
81 36        10
Do you want to insert again y/n
y
Enter the item you want to insert
76
81 36        10        76
Do you want to insert again y/n
y
Enter the item you want to insert
102
81 36        10        76        102
Do you want to insert again y/n
n
5  13
2
```

---

16. Write C/C++ program to implement stack with the use of array. Make a new function Multi Pop which pops k times. Take k as user input. Uses amortize analysis (accounting) to calculate time complexity of the program.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int *s;
int top=-1;
int max=100;
int cost=0;
void push(int item)
{
   if(top==max-1)
   {
      printf("Stack FULL\n");
   }
   else{
      s[++top]=item;
      printf("Top at :%d\n",top);
   }
   cost+=2;
}
int pop()
{
   if(top==-1)
   {
      printf("Underflow\n");
```

```
        return -1;
    }
    else{
        return s[top--];
    }
}
void multipop(int k)
{
    while(k<=(top+1) && k>0)
    {
        int p=pop();
        k--;
        printf("%d\n",p);
    }
}
int main()
{
    s=(int *)malloc(sizeof(int)*max);
    int k,item,size;char ch;
    do{
        printf("Enter the item you want to insert\n");
        scanf("%d",&item);
        push(item);
        printf("\n");
        printf("Do you want to insert again y/n\n");
        scanf("%c",&ch);
        scanf("%c",&ch);
    }while(ch=='y');
    size =(top+1);
    printf("Enter the value of K\n");
    scanf("%d",&k);
    printf("performing pop %d times\n",k);
    multipop(k);
    printf("total cost=%d\t size of input=%d\n",cost,size);
    printf("average cost=%d",(cost/size));
    return 0;
}
```

OUTPUT:

```
Enter the item you want to insert
24
Top at :0

Do you want to insert again y/n
```

```
y
Enter the item you want to insert
52
Top at :1

Do you want to insert again y/n
y
Enter the item you want to insert
1
Top at :2

Do you want to insert again y/n
y
Enter the item you want to insert
65
Top at :3

Do you want to insert again y/n
y
Enter the item you want to insert
16
Top at :4

Do you want to insert again y/n
n
Enter the value of K
3
performing pop 3 times
16
65
1
total cost=10     size of input=5
average cost=2
```

---

17. Write C/C++ program to implement **KMP** string matching method to find the pattern string in a text string both given by the user. Compute the complexity of the method for a text string of length N and pattern string of length M, where N>M.

Program:

```
#include <stdio.h>
#include <string.h>
#include<stdlib.h>

void compute_Pi(char* pat, int M, int* pi){
    int len = 0;
    pi[0] = 0;
    int i = 1;
    while (i < M) {
        if (pat[i] == pat[len]) {
            len++;
            pi[i] = len;
            i++;
        }
        else{
            if (len != 0) {
                len = pi[len - 1];
            }
            else{
                pi[i] = 0;
                i++;
            }
        }
    }
}
void KMP_Matcher(char* pat, char* txt){
    int M = strlen(pat);
    int N = strlen(txt);
    int pi[M];
    compute_Pi(pat, M, pi);
    int i = 0;
    int j = 0;
    while (i < N) {
        if (pat[j] == txt[i]) {
            j++;
            i++;
        }
```

```
        if (j == M) {
            printf("Found pattern at index %d \n", i - j);
            j = pi[j - 1];
        }
        else if (i < N && pat[j] != txt[i]) {
            if (j != 0)
                j = pi[j - 1];
            else
                i = i + 1;
        }
    }
}
int main(){
    char txt[] = "ABABDABACDABABCABAB";
    char pat[] = "ABABCABAB";
    printf("Text: %s\n", txt);
    printf("Pattern: %s\n", pat);
    KMP_Matcher(pat, txt);
    return 0;
}
```

OUTPUT:

Text: ABABDABACDABABCABAB

Pattern: ABABCABAB

Found pattern at index 10

Time and Space Complexity

Algorithm	**Time Complexity**			**Space complexity**
	Best case	Average case	Worst case	
1) Fibonacci series(iterative)	O(n)	O(n)	O(n)	O(1)
2) Fibonacci series (recursive)	O(n)	O(n)	O(n)	O(1)
3) Linear Search	O(1)	O(n)	O(n)	O(1)
4) Binary Search	O(1)	O(logn)	O(logn)	O(1)
5) Bubble Sort	O(n)	$O(n^2)$	$O(n^{2)}$	O(1)
6) Insertion Sort	O(n)	$O(n^2)$	$O(n^{2)}$	O(1)
7) Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)
8) Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
9) Quick Sort	O(nlogn)	O(nlogn)	$O(n^2)$	O(1)
10) Count Sort	O(n+k)	O(n+k)	O(n+k)	O(k+n)
11)Radix Sort	O(nd)	O(nd)	O(nd)	O(k+n)
12) Insert into heap	O(logn)	O(logn)	O(logn)	O(1)
13) Delete N elements from heap	O(nlogn)	O(nlogn)	O(nlogn)	O(1)
14) Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(1)
15) Dynamic Array	O(1)	O(n)	O(n)	O(n)
16) Multi pop	O(k)	O(k)	O(k)	O(1)
17) KMP string matching	O(m+n)	O(m+n)	O(m+n)	O(m)