1.

a)Implement Euclid's, consecutive integer checking and modified Euclid's algorithms to find GCD of two nonnegative integers.

```
#include<stdio.h>

#include<stdlib.h>

#define x 10

#define y 100


float euclid(int m,int n)

{

        int r;

        float count=0;

        while(n)

        {

                count++;

                r=m%n;

                m=n;

                n=r;

        }

        return count;

}


float consec(int m, int n)
```

```
{
        int min;
        float count=0;
        min=m;
        if(n<min)
                min=n;
        while(1)
        {
                count++;
                if(m%min==0)
                {
                        count++;
                        if(n%min==0)
                                break;
                        min-=1;
                }
                else
                        min-=1;
        }
        return count;
}

float modified(int m,int n)
```

```
{
        int temp;
        float count=0;
        while(n>0)
        {
                if(n>m)
                {
                        temp=m;m=n;n=temp;
                }
                m=m-n;
                count+=.5;
        }
        return count;
}

void analysis(int ch)
{
        int m,n,i,j,k;
        float count,maxcount,mincount;
        FILE *fp1,*fp2;
        for(i=x;i<=y;i+=10)
        {
                maxcount=0;
```

```
mincount=10000;
for(j=2;j<=i;j++)
{
        for(k=2;k<=i;k++)
        {
                count=0;
                m=j;
                n=k;
                switch(ch)
                {
                        case 1:count=euclid(m,n);
                                        break;
                        case 2:count=consec(m,n);
                                        break;
                        case 3:count=modified(m,n);
                                        break;
                }
                if(count>maxcount)
                        maxcount=count;
                if(count<mincount)
                        mincount=count;
        }
}
```

```
                switch(ch)
                {
                        case 1:fp2=fopen("e_b.txt","a");
                                        fp1=fopen("e_w.txt","a");
                                        break;
                        case 2:fp2=fopen("c_b.txt","a");
                                        fp1=fopen("c_w.txt","a");
                                        break;
                        case 3:fp2=fopen("m_b.txt","a");
                                        fp1=fopen("m_w.txt","a");
                                        break;
                }
                fprintf(fp2,"%d %.2f\n",i,mincount);
                fclose(fp2);
                fprintf(fp1,"%d %.2f\n",i,maxcount);
                fclose(fp1);
        }
}

int main()
{
        int ch;
        while(1)
```

```
        {
                printf("GCD\n");
                printf("1.Euclid\n2.modified Euclid\n3.consecutive integer method\n");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:
                        case 2:
                        case 3: analysis(ch);
                                        break;
                        default:exit(1);
                }
        }
        return 0;
}
```

Output:

Euclids best case and worst case

```
10 1.00
20 1.00
30 1.00
40 1.00
50 1.00
60 1.00
70 1.00
80 1.00
90 1.00
100 1.00
```

```
10 5.00
20 6.00
30 7.00
40 8.00
50 8.00
60 9.00
70 9.00
80 9.00
90 10.00
100 10.00
```

Consecutive integer method:

Best case:

```
10 2.00
20 2.00
30 2.00
40 2.00
50 2.00
60 2.00
70 2.00
80 2.00
90 2.00
100 2.00
```

Worst case:

```
10 12.00
20 24.00
30 36.00
40 46.00
50 58.00
60 70.00
70 76.00
80 88.00
90 100.00
100 108.00
```

Modified euclid method:

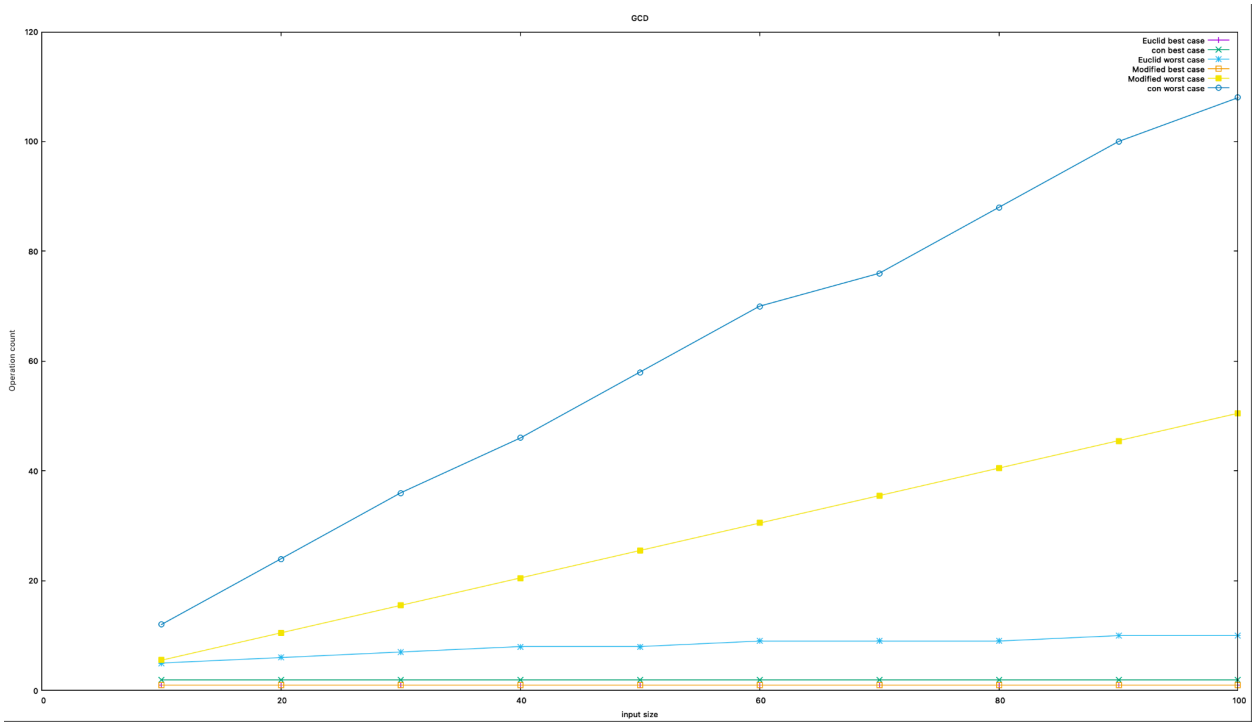
Best case:

```
10 1.00
20 1.00
30 1.00
40 1.00
50 1.00
60 1.00
70 1.00
80 1.00
90 1.00
100 1.00
```

Worst case:

```
10 5.50
20 10.50
30 15.50
40 20.50
50 25.50
60 30.50
70 35.50
80 40.50
90 45.50
100 50.50
```

Gnuplot:



b) Sequential search and Binary search algorithms for searching an element in the list.

```
#include<stdio.h>

#include<stdlib.h>


int search(int *a,int e,int key)

{

        int count=0;

        int s=0,mid;

        while(s<=e)

        {
```

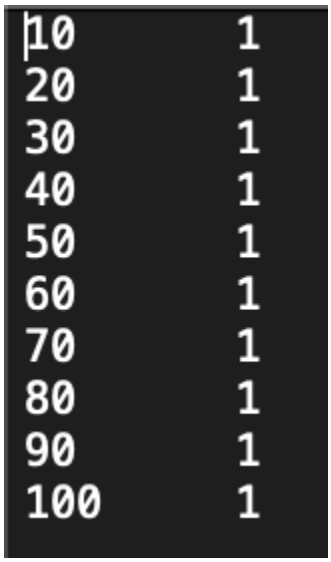
```
            mid=(s+e)/2;
            count++;
            if(a[mid]==key)
                  break;
            else if(a[mid]>key)
                  e=mid-1;
            else
                  s=mid+1;
      }
      return count;
}

void analysis(int ch)
{
      FILE * fp;
      int count;
      for(int i=10;i<=100;i+=10)
      {
            int key;
            int *a=(int *)malloc(i*sizeof(int));
            for(int j=0;j<i;j++)
                  a[j]=j;
            switch(ch)
```

```
		{
			case 1:fp=fopen("bsearch_b.txt","a");
				key=a[(i-1)/2];
				count=search(a,i-1,key);
				fprintf(fp,"%d\t%d\n",i,count);
				fclose(fp);
				break;
			case 2:fp=fopen("bsearch_w.txt","a");;
				key=a[0];
				count=search(a,i-1,key);
				fprintf(fp,"%d\t%d\n",i,count);
				fclose(fp);
				break;

		}
	}
}

int main()
{
	int ch;
	printf("1.Best case\n2.Worst case\n");
	scanf("%d",&ch);
```

```
        switch(ch)
        {
                case 1:
                case 2:
                        analysis(ch);
                                break;
                default:exit(0);
        }

        return 0;
}
```

Output:

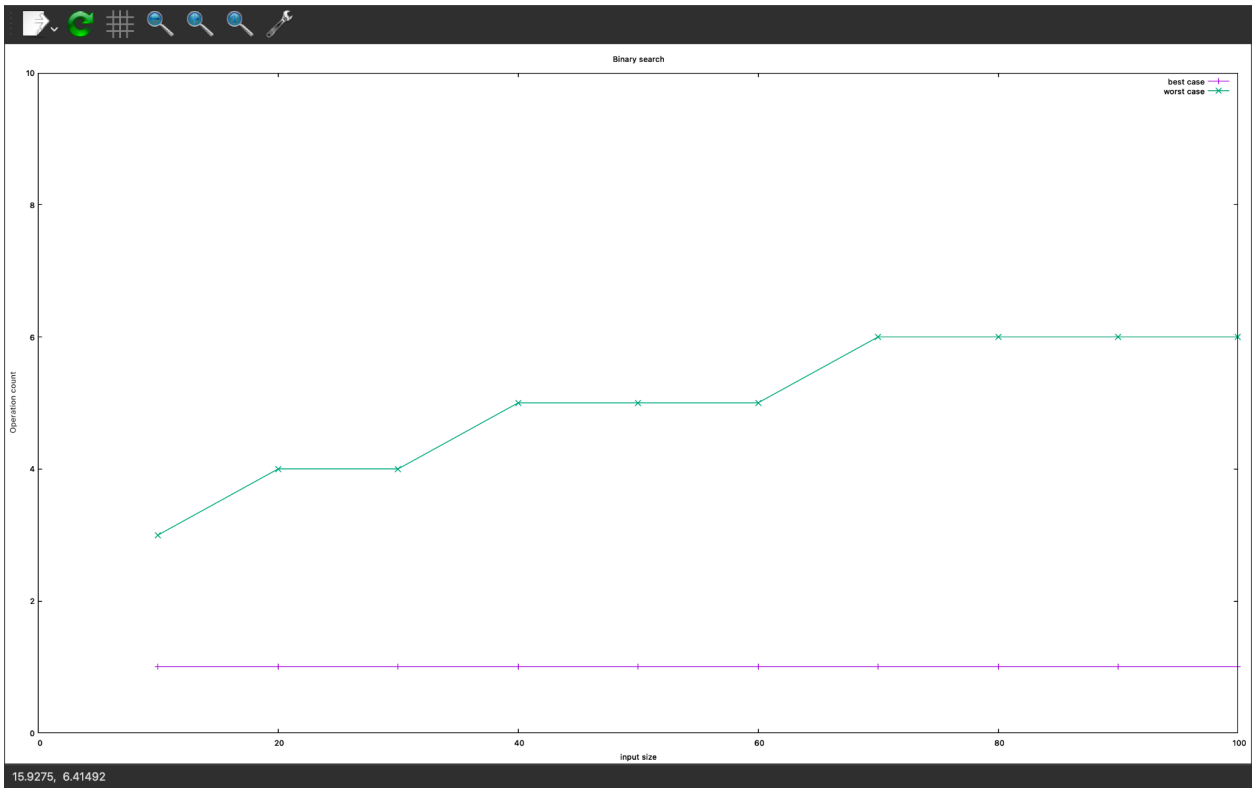
Best case:



Worst case:

10	3
20	4
30	4
40	5
50	5
60	5
70	6
80	6
90	6
100	6

Gnuplot:



2.

a) Implement Bubble sort algorithm to sort an array of ’N’ elements in descending order. Analyze the algorithm to find the order of growth of the algorithm’s basic operation count for best case and worst-case inputs and plot the graph for the same.

```
#include<stdio.h>

#include<stdlib.h>

#define x 10

#define y 100


int sort(int *b,int h)

{

        int flag,temp,count=0;

        for(int i=0;i<h-1;i++)

        {

                flag=0;

                for(int j=0;j<h-i-1;j++)

                {

                        count++;

                        if(b[j]>b[j+1])

                        {

                                temp=b[j];

                                b[j]=b[j+1];

                                b[j+1]=temp;
```

```
                        flag=1;
                }
        }
        if(flag==0)
                break;
    }
    return count;
}

void analysis(int ch)
{
    int *w,*b,count;
    FILE *fp1;
    for(int h=x;h<=y;h+=10)
    {
        count=0;
        w=(int*)malloc(h*sizeof(int));
        b=(int*)malloc(h*sizeof(int));
        for(int i=0;i<h;i++)
        {
            w[i]=h-i;
            b[i]=i;
        }
```

```
                switch(ch)
                {
                        case 1: count=sort(b,h);
                                        fp1=fopen("bsort_b.txt","a");
                                        fprintf(fp1,"%d\t",h);
                                        fprintf(fp1,"%d\n",count);
                                        fclose(fp1);
                                        break;
                        case 2: count=sort(w,h);
                                        fp1=fopen("bsort_w.txt","a");
                                        fprintf(fp1,"%d\t",h);
                                        fprintf(fp1,"%d\n",count);
                                        fclose(fp1);
                                        break;
                }
        }
        free(b);
        free(w);
}

int main()
{
        int ch;
```

```
        while(1)
        {
                printf("Enter the choice\n");
                printf("1.Best case analysis\n");
                printf("2.Worst case analysis\n");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:
                        case 2:analysis(ch);break;
                        default:exit(0);
                }
        }
        return 0;
}
```

Output:

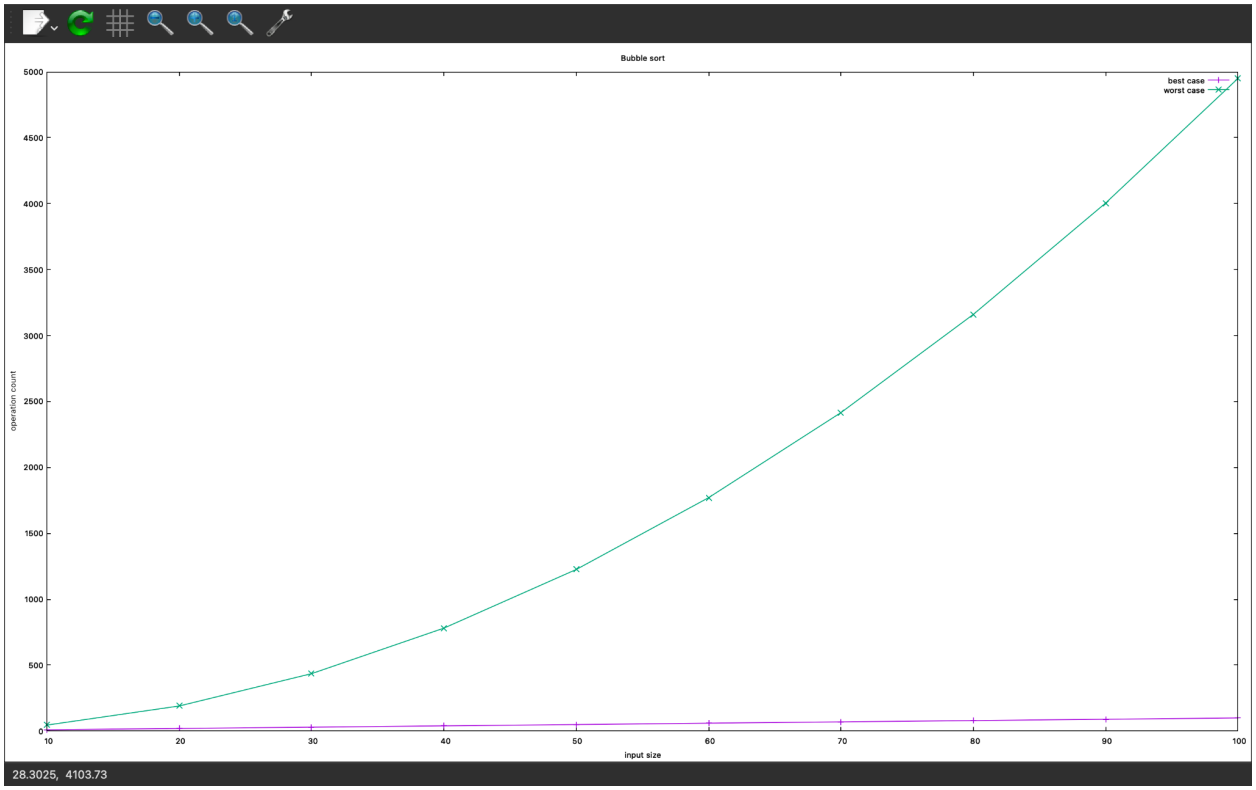
Best case:

10	9
20	19
30	29
40	39
50	49
60	59
70	69
80	79
90	89
100	99

Worst case:

10	45
20	190
30	435
40	780
50	1225
60	1770
70	2415
80	3160
90	4005
100	4950

Gnuplot:



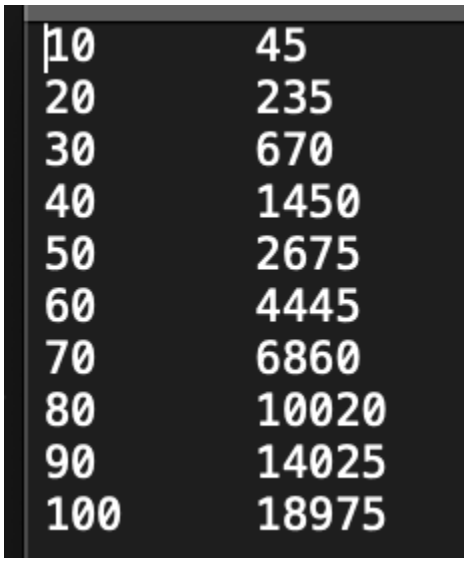
b) Implement Selection sort algorithm to sort an array of N elements in ascending order. Analyze the algorithm to find the order of growth of the algorithm's basic operation count for best case and worst-case inputs and plot the graph for the same.

```
#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#define x 10

#define y 100


int main()
```

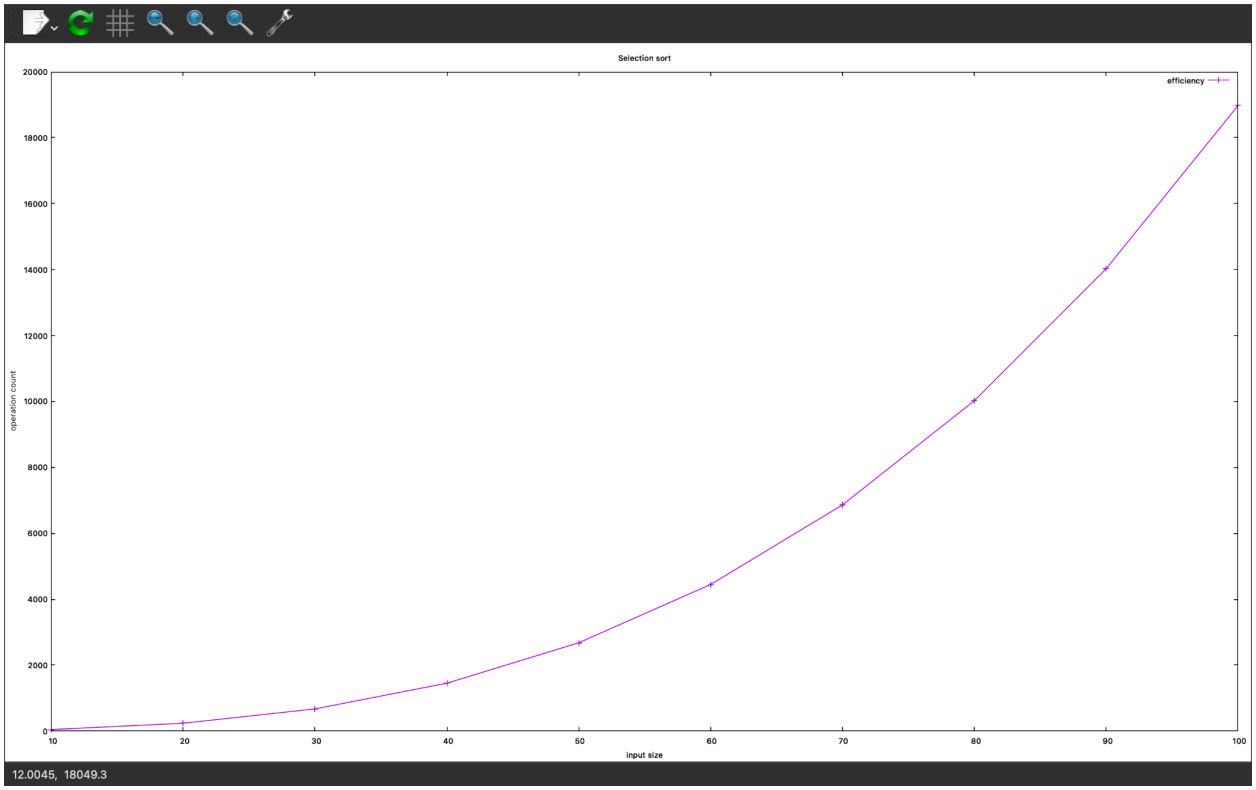
```
{
        int *a,count=0,min,temp;
        FILE *fp;
        fp=fopen("ssort.txt","a");
        srand(time(NULL));
        for(int h=x;h<=y;h+=10)
        {
                a=(int*)malloc(sizeof(int));
                for(int i=0;i<h;i++)
                        a[i]=rand()%100;
                for(int i=0;i<h-1;i++)
                {
                        min=i;
                        for(int j=i+1;j<h;j++)
                        {
                                count++;
                                if(a[j]<a[min])
                                        min=j;
                        }
                        temp=a[i];
                        a[i]=a[min];
                        a[min]=temp;
                }
```

```
            fprintf(fp,"%d\t%d\n",h,count);
        }
        fclose(fp);
        return 0;
}
```

Output:



Gnuplot:



3)

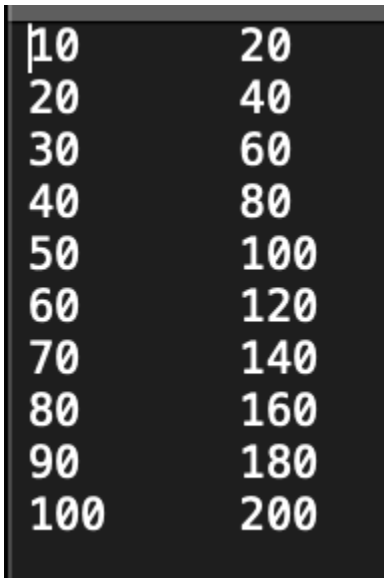
a) Applying suitable design strategy implement a linear algorithm to evaluate a given polynomial. Analyze the algorithm to find the order of growth of the algorithm's basic operation count and plot the graph for the same.

```
#include<stdio.h>

#include<stdlib.h>

#define x 10

#define y 100

#define X 4


int main()

{
```

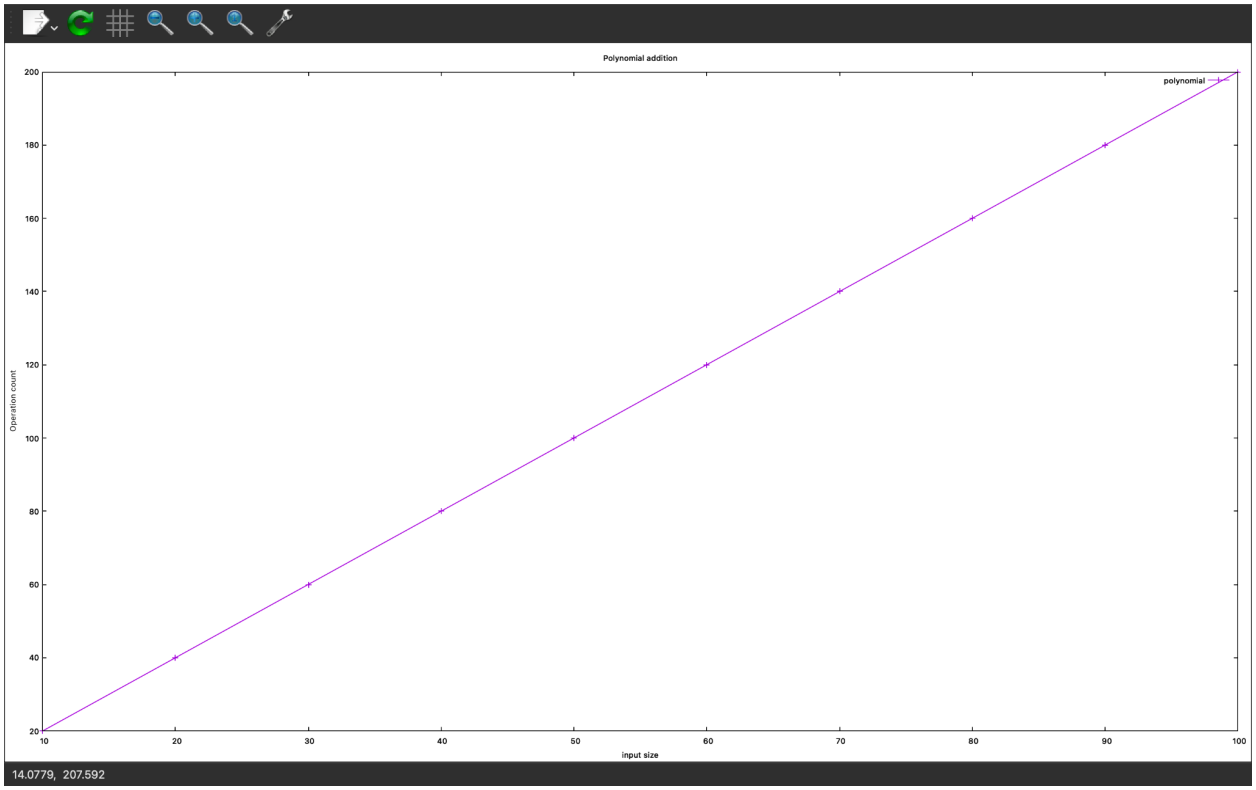
```
FILE *fp;
int count,px,poly;
fp=fopen("poly.txt","a");
for(int i=x;i<=y;i+=10)
{
        int *a=(int *)malloc(i*sizeof(int));
        count=0;
        for(int j=0;j<i;j++)
                a[j]=j;
        px=1;
        poly=a[0];
        for(int j=1;j<=i;j++)
        {
                count+=2;
                px=px*X;
                poly=poly+px*a[j];
        }
        fprintf(fp,"%d\t%d\n",i,count);
        free(a);
}
fclose(fp);
return 0;
```

```
}
```

Output:



Gnuplot:



b) Implement Brute force string matching algorithm to search for a pattern of length 'M' in a text of length 'N' (M<=N). Analyze the algorithm to find the order of growth of the algorithm's basic operation count for best case and worst-case inputs and plot the graph for the same

```
#include<stdio.h>

#include<stdlib.h>

#define x 10

#define y 10


int match(int *t,int *b,int i)

{

        int count=0;
```

```
        for(int j=0;j<=10-i;j++)
        {
                int k=0;
                count++;
                while(k<i && t[j+k]==b[k])
                {
                        count++;
                        k++;
                }
                if(k==i)
                        break;
        }
        return count;
}


void analysis(int ch)
{
        FILE *fp1;
        int *w,*b,*t,count;
        t=(int*)malloc(10*sizeof(int));
        for(int i=0;i<10;i++)
                t[i]=0;
```

```
for(int i=2;i<10;i++)
{
        w=(int*)malloc(i*sizeof(int));
        b=(int*)malloc(i*sizeof(int));
        for(int j=0;j<i;j++)
        {
                b[j]=0;
                if(j!=i-1)
                        w[j]=0;
                else
                        w[j]=1;
        }
        switch(ch)
        {
                case 1:count=match(t,b,i);
                        fp1=fopen("smatch_b.txt","a");
                        fprintf(fp1,"%d\t%d\n",i,count);
                        fclose(fp1);
                        break;
                case 2:count=match(t,w,i);
                        fp1=fopen("smatch_w.txt","a");
                        fprintf(fp1,"%d\t%d\n",i,count);
```

```
                                fclose(fp1);
                                break;
                        }
                }
}
int main()
{
        int ch;
        printf("Enter the choice\n");
        printf("1.Best case\n2.Worst case\n");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1:
                case 2:analysis(ch);break;
                default:exit(0);
        }
        return 0;
}
```

Output:

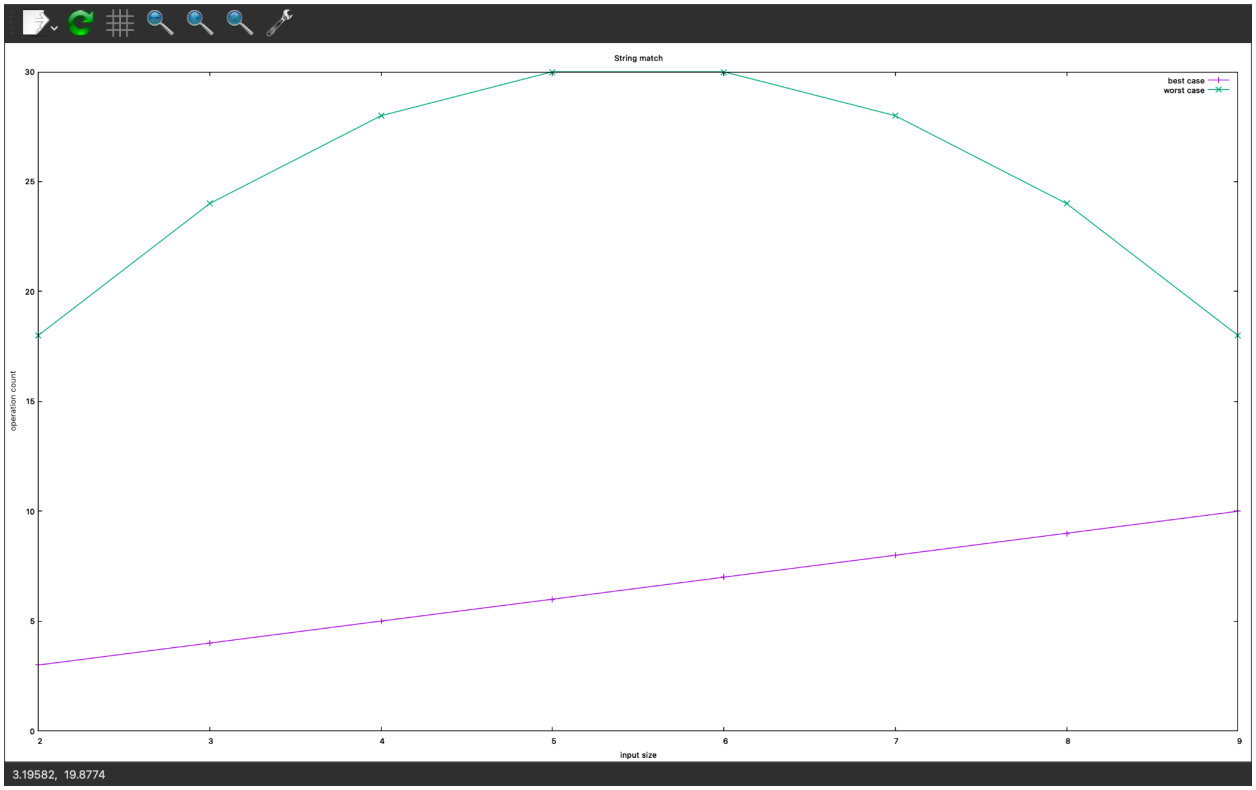
Best case:

2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	10

Worst case:

2	18
3	24
4	28
5	30
6	30
7	28
8	24
9	18

Gnuplot:



4)

a) Implement Merge sort algorithm to sort an array of 'N' elements in descending order. Analyze the algorithm to find the order of growth of the algorithm's basic operation count for best case and worst-case inputs and plot the graph for the same.

```
#include<stdio.h>

#include<stdlib.h>

int count=0;


void join(int arr1[], int left1[], int right1[],int l1, int m1, int r1){

   int i;

   for (i = 0; i <= m1 - l1; i++)
```

```
    arr1[i] = left1[i];
  for (int j = 0; j < r1 - m1; j++)
    arr1[i + j] = right1[j];
}

void split(int arr1[], int left1[], int right1[],int l1, int m1, int r1){
  for (int i = 0; i <= m1 - l1; i++)
    left1[i] = arr1[i * 2];
  for (int i = 0; i < r1 - m1; i++)
    right1[i] = arr1[i * 2 + 1];
}

void generateworstcase(int arr1[], int l1, int r1){
  if (l1 < r1){
    int m1 = l1 + (r1 - l1) / 2;
    int left1[m1 - l1 + 1];
    int right1[r1 - m1];
    split(arr1, left1, right1, l1, m1, r1);
    generateworstcase(left1, l1, m1);
    generateworstcase(right1, m1 + 1, r1);
    join(arr1, left1, right1, l1, m1, r1);
  }
}
```

```
void merge(int *a,int l,int mid,int h)
{
	int i,j,k;
	i=l;j=mid+1;k=l;
	int b[101];
	while(i<=mid && j<=h)
	{
		count++;
		if(a[i]<a[j])
			b[k++]=a[i++];
		else
			b[k++]=a[j++];
	}
	for(;i<=mid;i++)//here we are using <= bcause mid and h are indicies
		b[k++]=a[i];
	for(;j<=h;j++)
		b[k++]=a[j];
	for(i=l;i<=h;i++)
		a[i]=b[i];
}

void msort(int *a,int l,int h)
```

```
{

        int i=l,j=h,mid;
        if(i<j)
        {
                mid=(l+h)/2;
                msort(a,l,mid);
                msort(a,mid+1,h);
                merge(a,l,mid,h);
        }
}

void analysis(int ch)
{
        FILE *fp;
        int *w,*b;
        for(int i=10;i<=100;i+=10)
        {
                //w=(int*)malloc(i*sizeof(int));
                b=(int*)malloc(i*sizeof(int));
                w=(int*)malloc(i*sizeof(int));
                for(int j=0;j<i;j++)
                {
```

```
                b[j]=j;
                w[j]=j;
            }
            switch(ch)
            {
                case 1:msort(b,0,i-1);
                        fp=fopen("msort_b.txt","a");
                        fprintf(fp,"%d\t%d\n",i,count);
                        count=0;
                        fclose(fp);
                        break;
               case 2:generateworstcase(w,0,i-1);
                     msort(w,0,i-1);
                        fp=fopen("msort_w.txt","a");
                        fprintf(fp,"%d\t%d\n",i,count);
                        count=0;
                        fclose(fp);
                        break;

            }
        }
}
```

```
int main()
{
	int ch;
	printf("1.Best case\n2.Worst case\nEnter your choise\n");
	scanf("%d",&ch);
	switch(ch)
	{
		case 1:
		case 2:
				analysis(ch);
				break;
		default:exit(0);
	}
	return 0;
}
```

Output:

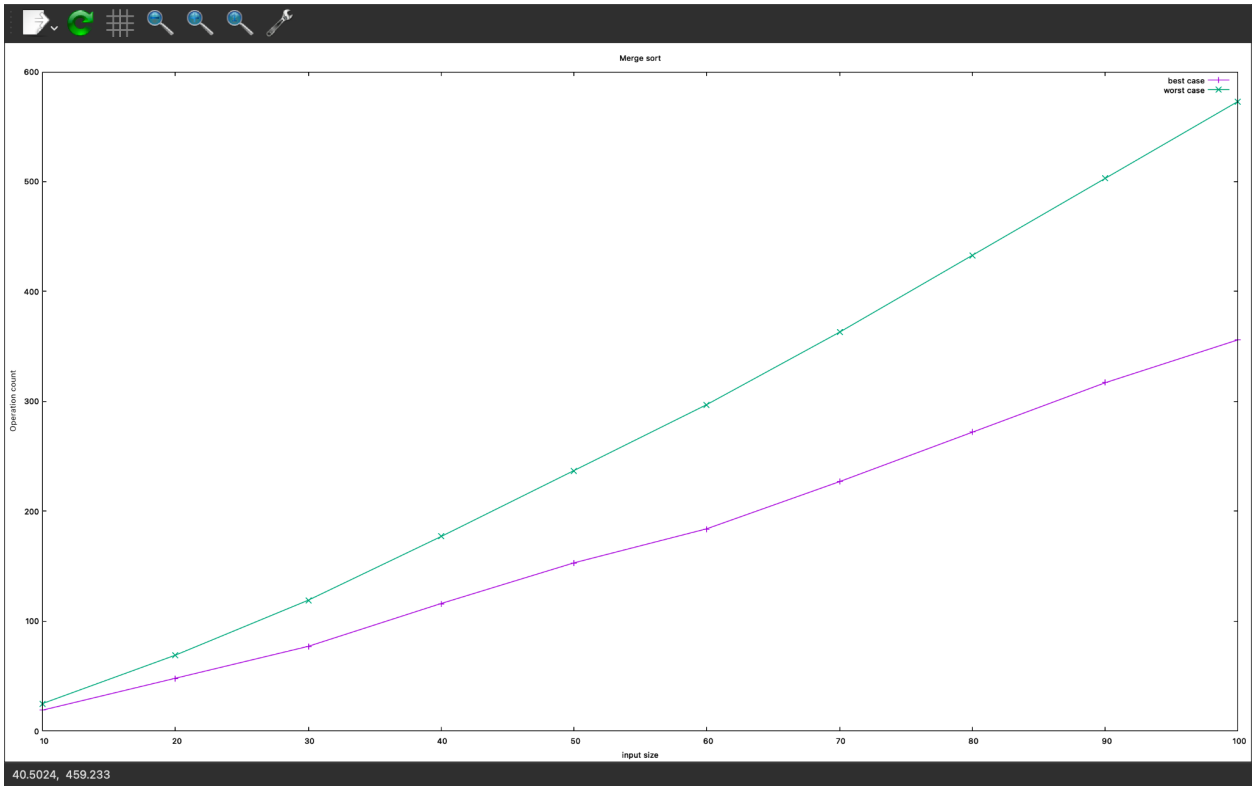
Best case:

```
10      19
20      48
30      77
40      116
50      153
60      184
70      227
80      272
90      317
100     356
```

Worst case:

```
10      25
20      69
30      119
40      177
50      237
60      297
70      363
80      433
90      503
100     573
```

Gnuplot:



b) Implement Quick sort algorithm to sort an array of 'N' elements in descending order. Analyze the algorithm to find the order of growth of the algorithm's basic operation count for best case and worst-case inputs and plot the graph for the same

```
#include<stdio.h>

#include<stdlib.h>

int count=0;

int partition(int a[],int l,int r)

{

 int p=a[l];

 int i=l+1;

 int j=r;
```

```
while(i<=j)
{
 count++;
 while(i<=r && a[i]<p)
 {
  i++;
  count++;
 }
 if(i>r)count--;
 count++;
 while(a[j]>p)
 {
  j--;
  count++;
 }
if(i<=j)
{
 int temp=a[i];
 a[i]=a[j];
 a[j]=temp;
 i++;
 j--;
}
```

```
 }
 int temp=a[j];
 a[j]=a[l];
 a[l]=temp;
 return j;
}
void qksort(int *a,int l,int r)
{
        int s;
 if(l<r)
 {
  s=partition(a,l,r);
  qksort(a,l,s-1);
  qksort(a,s+1,r);
 }
}




void analysis(int ch)
{
        FILE *fp;
        int *b,*w;
```

```
for(int i=10;i<=100;i+=10)
{
        w=(int*)malloc(i*sizeof(int));
        b=(int*)malloc(i*sizeof(int));
        for(int j=0;j<i;j++)
        {
                b[j]=i;
                w[j]=j;
        }
        switch(ch)
        {
                case 1:fp=fopen("qsort_b.txt","a");
                        qksort(b,0,i-1);
                        fprintf(fp,"%d\t%d\n",i,count);
                        count=0;
                        fclose(fp);
                        free(b);
                        break;
                case 2:fp=fopen("qsort_w.txt","a");
                        qksort(w,0,i-1);
                        fprintf(fp,"%d\t%d\n",i,count);
                        count=0;
                        fclose(fp);
```

```
                        free(w);
                        break;
                }
        }
}

int main()
{
        int ch;
        printf("1.Best case\n2.Worst case\nEnter your choice\n");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1:
                case 2:analysis(ch);
                        break;
                default:exit(1);
        }
        return 0;
}
```

Output:

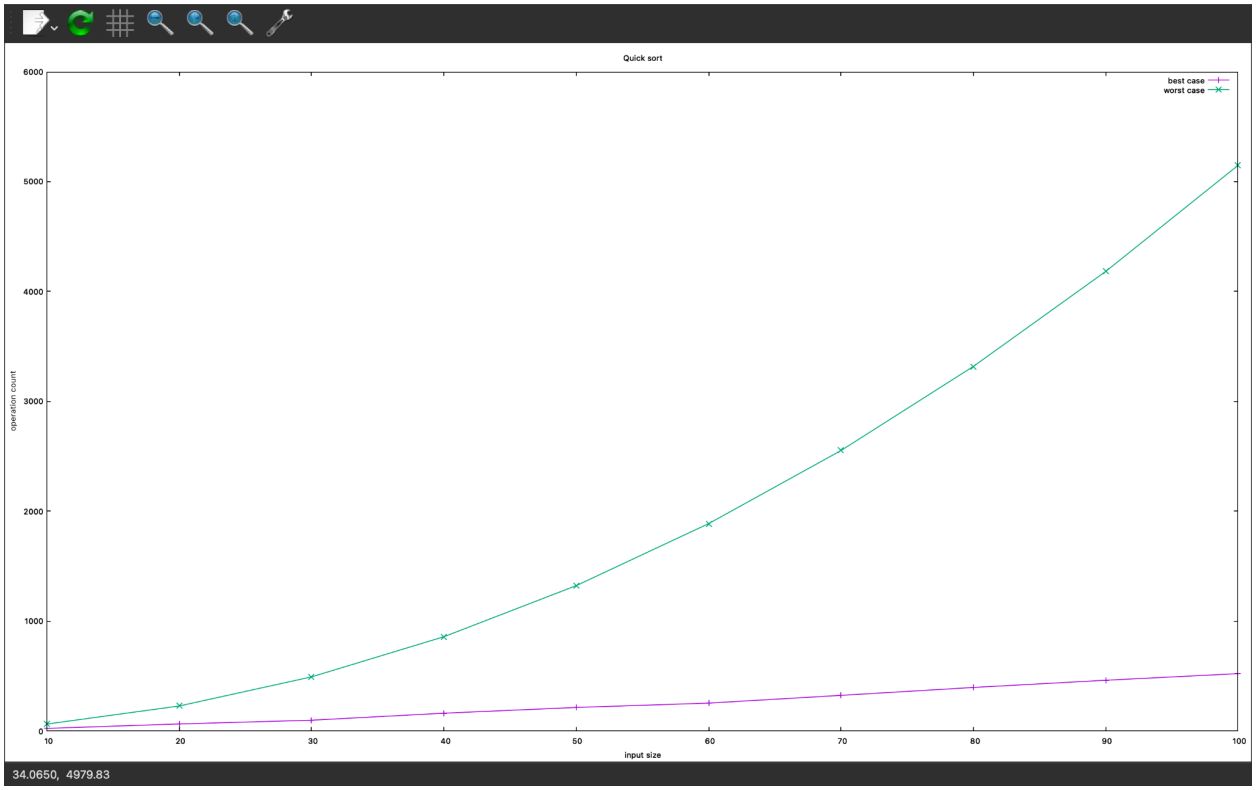
Best case:

10	24
20	64
30	98
40	162
50	214
60	254
70	324
80	396
90	462
100	522

Worst case:

10	63
20	228
30	493
40	858
50	1323
60	1888
70	2553
80	3318
90	4183
100	5148

Gnuplot:



5)

a) Implement Recursive binary search algorithm to search for an element in an array of N elements. Analyze the algorithm to find the order of growth of the algorithm's basic operation count for best case and worst-case inputs and plot the graph for the same. Also count and establish the order of growth of function calls.

```
#include<stdio.h>

#include<stdlib.h>


int count=0;

int rsearch(int *a,int s,int e,int key)

{
```

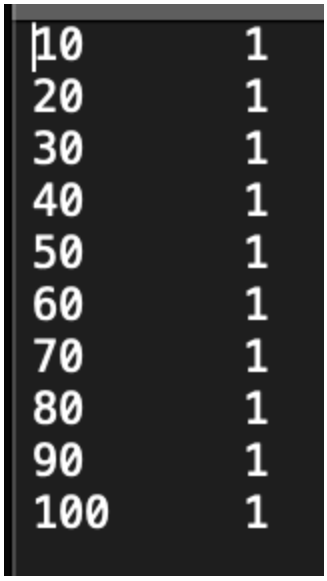
```
        int mid;
        if(s>e)
                return count;

        {
                mid=(s+e)/2;
                count++;
                if(a[mid]==key)
                        return count;
                else if(a[mid]>key)
                        return rsearch(a,s,mid-1,key);
                else
                        return rsearch(a,mid+1,e,key);
        }
}


void analysis(int ch)
{
        FILE * fp;
        //int count;
        for(int i=10;i<=100;i+=10)
        {
```

```
            int key;
            int *a=(int *)malloc(i*sizeof(int));
            for(int j=0;j<i;j++)
                a[j]=j;
            switch(ch)
            {
                case 1:fp=fopen("rbsearch_b.txt","a");
                        key=a[(i-1)/2];
                        count=0;
                        rsearch(a,0,i-1,key);
                        fprintf(fp,"%d\t%d\n",i,count);
                        fclose(fp);
                        break;
                case 2:fp=fopen("rbsearch_w.txt","a");
                        key=a[0];
                        rsearch(a,0,i-1,key);
                        fprintf(fp,"%d\t%d\n",i,count);
                        count=0;
                        fclose(fp);
                        break;
            }
        }
}
```

```
int main()
{
        int ch;
        printf("1.Best case\n2.Worst case\n");
        scanf("%d",&ch);

        switch(ch)
        {
                case 1:
                case 2:
                        analysis(ch);
                                break;
                default:exit(0);
        }

        return 0;
}
```

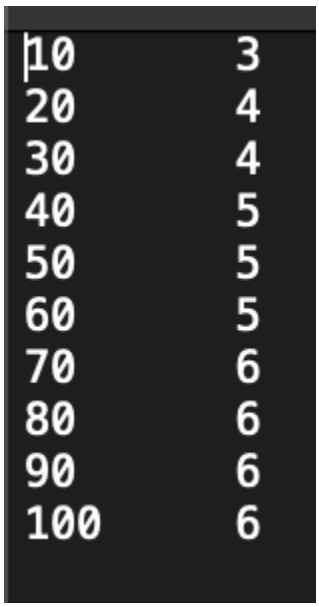
Output:

Best case:



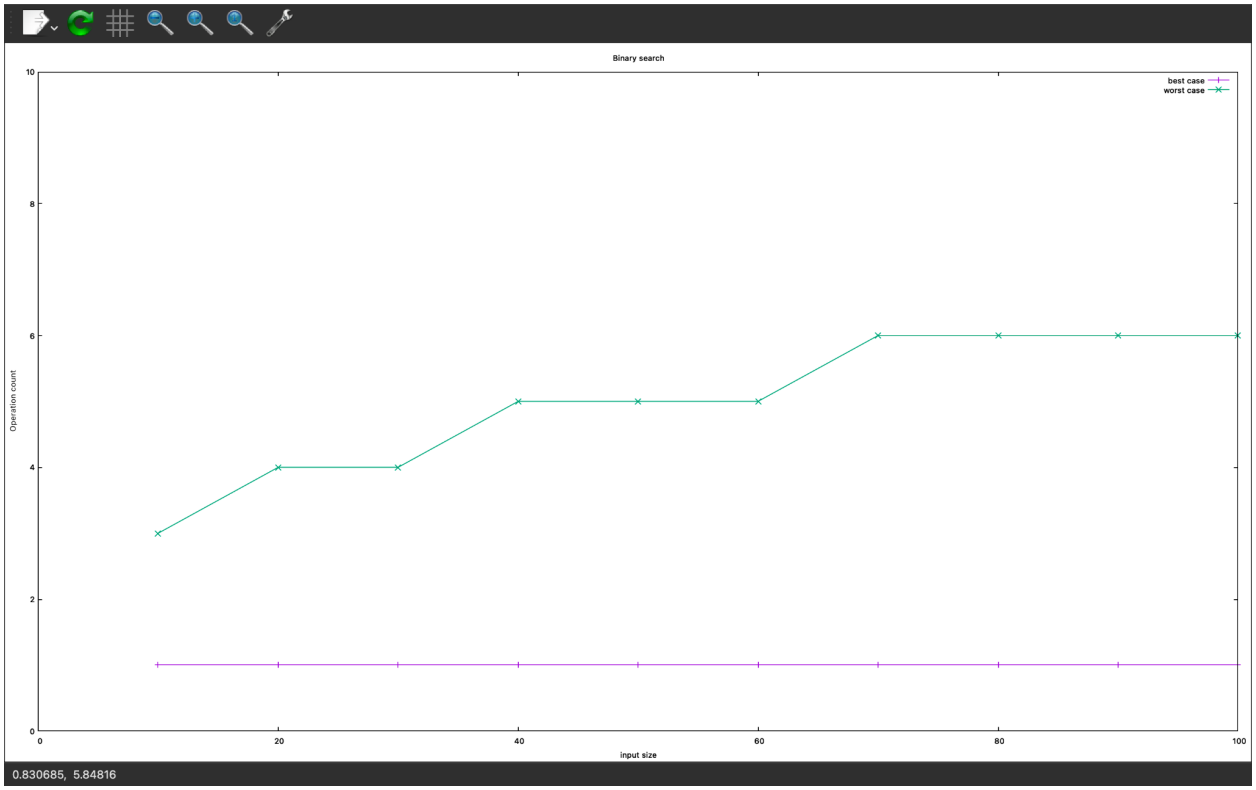
10	1
20	1
30	1
40	1
50	1
60	1
70	1
80	1
90	1
100	1

Worst case:



10	3
20	4
30	4
40	5
50	5
60	5
70	6
80	6
90	6
100	6

Gnuplot:



b) Applying divide and conquer technique, implement an algorithm to find largest and smallest element in an array of 'N' elements. Analyze the algorithm to find the order of growth of the algorithm's basic operation count andplot the graph for the same.

```
#include<stdio.h>

#include<stdlib.h>

#include<time.h>


int count=0;


int maxmin(int *a,int s,int e)

{
```

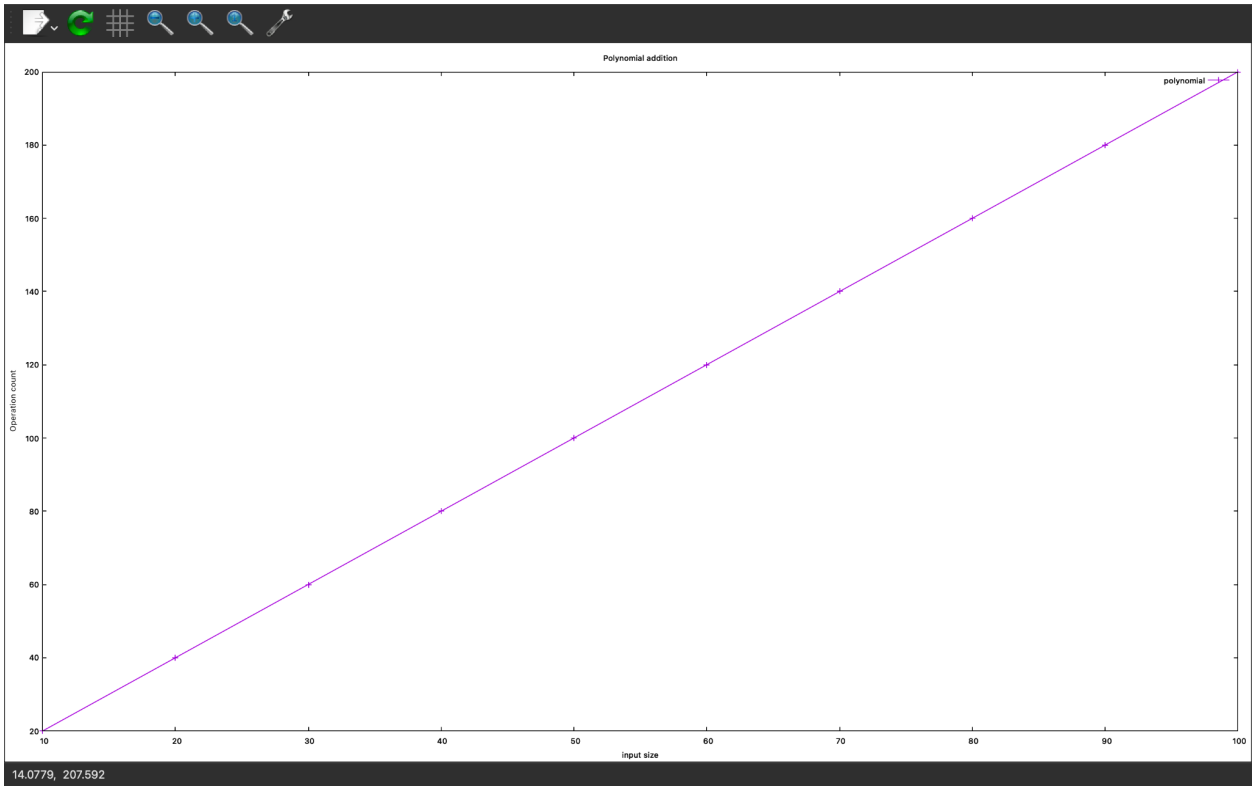
```
	int t1,t2;
	if(s==e)
		return a[s];
	t1=maxmin(a,s,(s+e)/2);
	t2=maxmin(a,(s+e)/2+1,e);
	count++;

	if(t1<t2)
		return t1;
	else
		return t2;
}

int main()
{
	int *a;
	FILE *fp;
	srand(time(NULL));

	for(int i=10;i<=100;i+=10)
	{
		a=(int*)malloc(sizeof(int)*i);
		for(int j=0;j<i;j++)
```

```
                a[j]=rand()%100;
            fp=fopen("maxmin.txt","a");
            maxmin(a,0,i-1);
            fprintf(fp,"%d\t%d\n",i,count);
            count=0;
            fclose(fp);
        }
        return 0;
}
```

Output:

```
10      9
20      19
30      29
40      39
50      49
60      59
70      69
80      79
90      89
100     99
```

Gnuplot:



6)

a) Implement Insertion sort algorithm, analyze its efficiency for worst case and best- case inputs (specify worst case and best-case inputs) and plot the graph

```
#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#define x 10

#define y 100


int isort(int *b,int h)

{
```

```
        int v,j,count=0;
        for(int i=1;i<h;i++)
        {
                v=b[i];
                j=i-1;
                count++;
                while(j>=0 && b[j]>v)
                {
                        count++;
                        b[j+1]=b[j];
                        j=j-1;
                }
                if(j<0)count--;
                b[j+1]=v;
        }
        return count;
}

void analysis(int ch)
{
        int *w,*b,*a,count;
        FILE *fp1;
        srand(time(NULL));
```

```
for(int h=x;h<=y;h+=10)
{
        count=0;
        w=(int*)malloc(h*sizeof(int));
        b=(int*)malloc(h*sizeof(int));
        a=(int*)malloc(h*sizeof(int));
        for(int i=0;i<h;i++)
        {
                w[i]=h-i;
                b[i]=i;
                a[i]=rand()%100;
        }
        switch(ch)
        {
                case 1: count=isort(b,h);
                                fp1=fopen("isort_b.txt","a");
                                fprintf(fp1,"%d\t",h);
                                fprintf(fp1,"%d\n",count);
                                fclose(fp1);
                                break;
                case 2: count=isort(w,h);
                                fp1=fopen("isort_w.txt","a");
                                fprintf(fp1,"%d\t",h);
```

```
                                        fprintf(fp1,"%d\n",count);
                                        fclose(fp1);
                                        break;
                        case 3: count=isort(a,h);
                                        fp1=fopen("isort_a.txt","a");
                                        fprintf(fp1,"%d\t",h);
                                        fprintf(fp1,"%d\n",count);
                                        fclose(fp1);
                                        break;
                }
        }
        free(b);
        free(w);
}

int main()
{
        int ch;
        while(1)
        {
                printf("Enter the choice\n");
                printf("1.Best case analysis\n");
                printf("2.Worst case analysis\n");
```

```
            printf("3.Average case analysis\n");
            scanf("%d",&ch);
            switch(ch)
            {
                  case 1:
                  case 2:
                  case 3:analysis(ch);break;
                  default:exit(0);
            }
      }
      return 0;
}
```

Output:

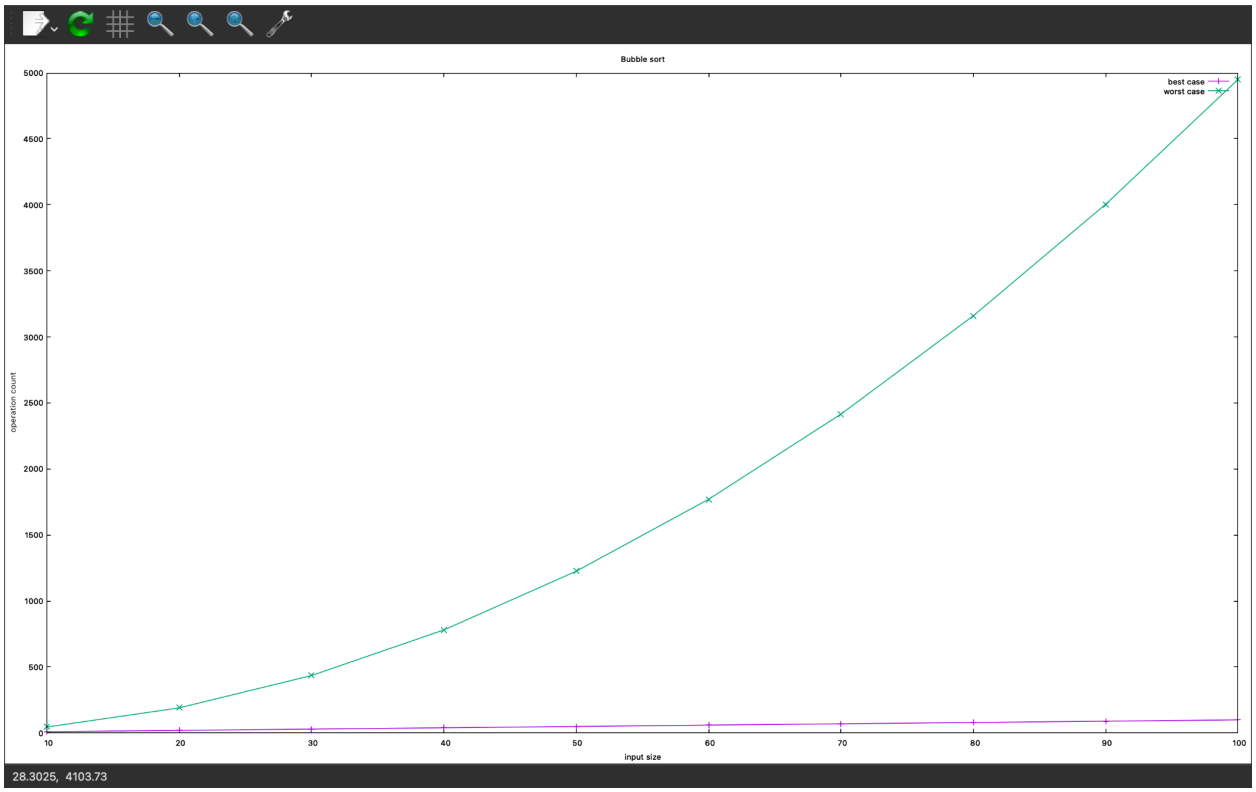
Best case:

```
10       9
20       19
30       29
40       39
50       49
60       59
70       69
80       79
90       89
100      99
```

Worst case:

```
10          45
20          190
30          435
40          780
50          1225
60          1770
70          2415
80          3160
90          4005
100         4950
```

Gnuplot:



b) Implement DFS algorithm to check for connectivity of a graph. If not connected, display the connected components.

#include<stdio.h>

```
#include<stdlib.h>
int a[10][10];
int visited[10];
int con;
int cyc;
int count=0;
int connected[10];
int cyclic[10];
int cycle=0;
int bop=0;

void dfs(v,V)
{
   count+=1;
   visited[v]=count;
   connected[con++]=v+1;
   for(int w=0;w<V;w++)
   {
      if(a[v][w]==1 && visited[w]==1 && (visited[v]-visited[w]!=1))
      {
         cycle++;
      }
      bop++;
```

```
        if(a[v][w]==1)
        {

            if(visited[w]==0)
            {

                dfs(w,V);
            }
        }
    }
}

void DFS(int n)
{
    int flag;
    for(int i=0;i<n;i++)
    {
        flag=1;
        if(visited[i]==0)
        {
            con=0;
            dfs(i,n);
            flag=0;
```

```
        }
        if(con<n && flag==0)
        {
            printf("Graph is disconnected and the connected components are:\n");
            for(int j=0;j<con;j++)
                printf("%d ",connected[j]);
            printf("\n");
        }
    }
}

int main()
{
    int n,i,j;
    printf("Enter the number of vertiecies\n");
    scanf("%d",&n);
    printf("Enter the adjacency matrix\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%d",&a[i][j]);
    for(i=0;i<n;i++)
        visited[i]=0;
    DFS(n);
```

```
    printf("The order in which the vertiecies are visited are\n");
    for(i=0;i<n;i++)
        printf("vertex %d is visited-%d\n",i+1,visited[i]);
    printf("count:%d\n",bop);
    if(cycle>0)
        printf("Cycle exists\n");
    return 0;
}
```

Output:

```
Enter the number of vertiecies
4
Enter the adjacency matrix
0 1 1 0
1 0 1 0
1 1 0 0
0 0 0 0
Graph is disconnected and the connected components are:
1 2 3
Graph is disconnected and the connected components are:
4
The order in which the vertiecies are visited are
vertex 1 is visited-1
vertex 2 is visited-2
vertex 3 is visited-3
vertex 4 is visited-4
count:16
Cycle exists
```

7.

a. Implement BFS algorithm to check for connectivity of a graph. If not connected, display the connected components.

Program:

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

typedef struct node{
        int v;
        int flag;
}n;

typedef struct queue{
        n items[100];
        int f;
        int r;
}Q;

Q q;
int V[100],cyclic,connected,count;

void dequeue(){
        q.f++;
}

void enqueue(int v){
        q.r = (q.r+1);
```

```
        q.items[q.r].v = v;
        q.items[q.r].flag = 0;
}

void dfs(int *a[],int v,int n){
        count++;
        V[v] = count;
        printf("%d - %d\n",v,count);
        enqueue(v);
        while(q.f<=q.r){
                q.items[q.f].flag = 1;
                for(int i = 0;i<n;i++)
                        if(a[q.items[q.f].v][i] == 1){
                                if(V[i]==0){
                                        count++;
                                        V[i] = count;
                                        printf("%d - %d\n",i,count);
                                        enqueue(i);
                                }else if(V[v] < V[i])
                                        cyclic++;
                                // }else if(q.items[q.items[i].v].flag == 0)
                                //      cyclic++;
                        }
                dequeue();
        }
}

void BFS(int *a[],int n){
        for(int i = 0;i<n;i++)
                V[i] = 0;
        cyclic = 0;
        connected = -1;
        count = 0;
        q.r = -1;
```

```
        q.f = 0;
        for(int i = 0;i<n;i++)
                if(V[i] == 0){
                        dfs(a,i,n);
                        connected++;
                }
        if(cyclic>0)
                printf("Cyclic\n");
}
int main(){
        int v;
        q.f = 0;
        q.r = -1;
        printf("Enter the number of vertices\n");
                scanf("%d",&v);
        int *a[v];
        for(int i = 0;i<v;i++){
                a[i] = (int *)malloc(sizeof(int)*v);
                q.items[i].flag = -1;
        }
        printf("Enter the adjacency matrix\n");
        for(int i = 0;i<v;i++)
                for(int j = 0;j<v;j++)
                        scanf("%d",&a[i][j]);
        BFS(a,v);
}
```

Output:

```
Enter the number of vertices
4
Enter the adjacency matrix
0 1 1 0
1 0 1 0
1 1 0 1
0 0 1 0
Tree 1
0 - 1
1 - 2
2 - 3
3 - 4
Cyclic
```

b. Apply the DFS algorithm to perform topological sorting.

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define initial 1
#define visited 2
#define finished 3

int state[100];
int count,bop,cyclic,V[100],top[100];


void dfs(int v,int *a[100],int n){
        V[v] = -9;
        state[v] = visited;
        //cycle(v,a,n);
        for(int i = 0;i<n;i++)
                if(a[v][i] == 1){
                        if(state[i]==initial){
                                dfs(i,a,n);
                        }else if(state[i] == visited){
                                printf("Cycles exists hence no solution\n");
                                exit(1);
                        }
```

```
                }
        count++;
        V[v] = count;
        state[v] = finished;
        top[count-1] = v;
}

void DFS(int *a[100],int n){
        for(int i = 0;i<n;i++){
                state[i] = initial;
                V[i] = 0;
        }

        count = 0;
        for(int i = 0;i<n;i++)
                if(state[i]==initial)
                        dfs(i,a,n);
}

int main(){
        int v;
        printf("Enter the number of vertices\n");
                scanf("%d",&v);
        int *a[v];
        for(int i = 0;i<v;i++)
                a[i] = (int *)malloc(sizeof(int)*i);
        printf("Enter the adjacency matrix\n");
        for(int i = 0;i<v;i++)
                for(int j = 0;j<v;j++)
                        scanf("%d",&a[i][j]);
        DFS(a,v);
        printf("Topological sorting\n");
        for(int i = v-1;i>=0;i--)
                printf("%d ",top[i]);
}
```

Output:

```
Enter the number of vertices
4
Enter the adjacency matrix
0 1 1 0
0 0 1 0
0 0 0 1
0 0 0 0
Topological sorting
0 1 2 3 %
```

8. Implement source removal algorithm to solve topological sorting problem.

Program:

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

typedef struct queue{
        int items[100];
        int f,r;
        int count;
}Q;

//int inq[100];
Q q;

void indegree(int *a[100],int v,int *inq,int *flag){
        for(int i = 0;i<v;i++){
                for(int j = 0;j<v;j++)
                        if(a[j][i] == 1)inq[i] = inq[i]+1;
                if(inq[i]==0){
                        q.r = (q.r+1)%v;
                        q.items[q.r] = i;
                        q.count++;
                        flag[i] = 1;
                }
        }
        if(q.count==0){
                printf("Cycle exist hence no solution\n");
                exit(1);
```

```
        }
}

void source(int *a[100],int v,int *inq,int *flag){
        while(q.count != 0){
                int source = q.items[q.f];
                q.f = (q.f+1)%v;
                q.count--;
                printf("%d ",source);
                for(int i = 0;i<v;i++)
                {
                        if(a[source][i]==1)inq[i]--;
                        if(inq[i]==0 && flag[i]==0){
                                q.r = (q.r+1)%v;
                                q.items[q.r] = i;
                                q.count++;
                                flag[i] = 1;
                        }
                }
        }
}

int main(){
        int v;
        q.f = 0;
        q.r = -1;
        printf("Enter the number of vertices\n");
                scanf("%d",&v);
        int *a[v];
        for(int i = 0;i<v;i++)
                a[i] = (int *)malloc(sizeof(int)*v);
        printf("Enter the adjacency matrix\n");
                for(int i = 0;i<v;i++)
                        for(int j = 0;j<v;j++)
                                scanf("%d",&a[i][j]);
        int *inq = (int *)calloc(sizeof(int),v);
        int *flag = (int *)calloc(sizeof(int),v);
        indegree(a,v,inq,flag);
        printf("Toplological sorting\n");
                source(a,v,inq,flag);
```

}

Output:

```
Enter the number of vertices
4
Enter the adjacency matrix
0 1 1 0
0 0 1 0
0 0 0 1
0 0 0 0
Toplological sorting
0 1 2 3 %
```

9. Implement the heap sort algorithm with bottom-up heap construction. Analyze its efficiency.

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int count;
int delcount;

void Hdel(int *a,int n){
        int size = n;
        for(int i = 0;i<size-1;i++){
                int temp = a[1];
                a[1] = a[n];
                a[n] = temp;
                n--;
                int k = 1;
                int v = a[k];
                int heap = 0;
                while(!heap && 2*k<=n){
                        int j = 2*k;
                        if(j<n){
                                count++;
```

```
                        //delcount++;
                        if(a[j+1]>a[j])
                                j++;
                }
                count++;
                //delcount++;
                if(v>=a[j])
                        heap = 1;
                else{
                        a[k] = a[j];
                        k = j;
                }
        }
        a[k] = v;
    }
}
void Hin(int *a,int n){
    for(int i = n/2;i>=1;i--){
        int v = a[i];
        int k = i;
        int heap = 0;
        //printf("Hi\n");
        while(!heap && 2*k<=n){
                int j = 2*k;
                if(j<n){
                        count++;
                        //delcount++;
                        if(a[j+1]>a[j])
                        {
                                j++;
                        }
                }
                count++;
                //delcount++;
                if(v>=a[j])
                        heap = 1;
                else{
                        a[k] = a[j];
                        k = j;
                }
```

```
                }
                a[k] = v;
        }
}

void analysis(int ch){
        FILE *f;
        int *w,*b;
        for(int i = 10;i<=100;i+=10){
                b = (int *)malloc(sizeof(int)*(i+1));
                w = (int *)malloc(sizeof(int)*(i+1));
                for(int j = 1;j<i;j++){
                        b[j] = i-j;
                        w[j] = j;
                }
                count = 0;
                //printf("Hi\n");
                switch(ch){
                        case 1:
                                Hin(b,i-1);
                                f = fopen("conb.txt","a");
                                break;
                        case 2:
                                Hin(w,i-1);
                                f = fopen("conw.txt","a");
                                break;
                        case 3:
                                Hin(w,i-1);
                                Hdel(w,i-1);
                                f = fopen("del.txt","a");
                                break;
                }
                fprintf(f,"%d %d\n",i-1,count);
                fclose(f);
        }
}

int main(){
        int ch;
        while(1){
```

```
                printf("1.Best case construction\n2.Worst case
construction\n3.Heap sort\n");
                        scanf("%d",&ch);
                switch(ch){
                        case 1:
                        case 2:
                        case 3:
                                analysis(ch);break;
                        default: exit(1);
                }
        }
}
```

Output:

```
1.Best case construction
2.Worst case construction
3.Heap sort
1
1.Best case construction
2.Worst case construction
3.Heap sort
2
1.Best case construction
2.Worst case construction
3.Heap sort
3
1.Best case construction
2.Worst case construction
3.Heap sort
4
```

Heap construction worst case:

```
9 14
19 32
29 48
39 70
49 88
59 108
69 130
79 148
89 166
99 190
```

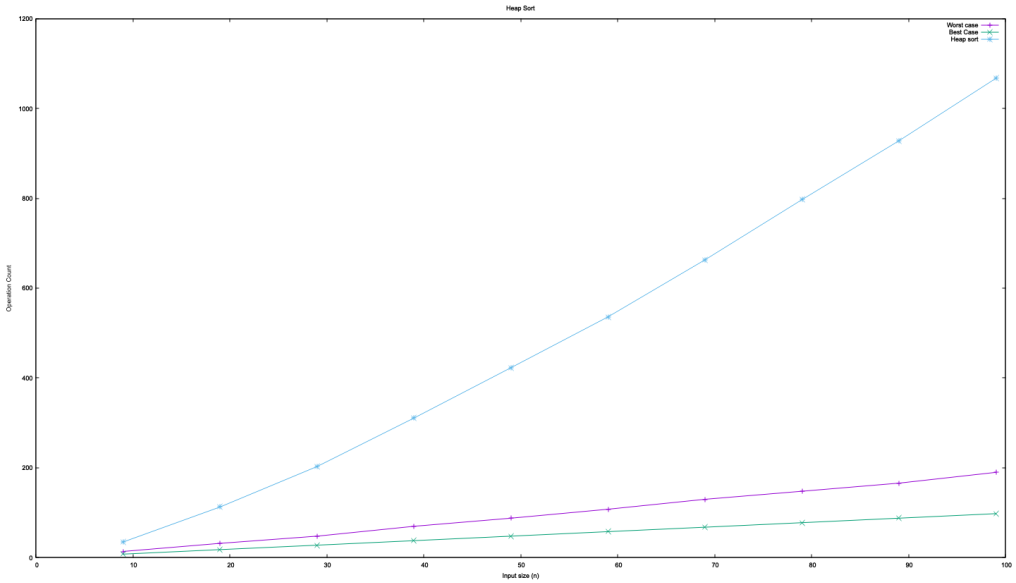
Heap construction best case:

```
9 8
19 18
29 28
39 38
49 48
59 58
69 68
79 78
89 88
99 98
```

Heap sort:

```
9 35
19 113
29 203
39 311
49 423
59 536
69 663
79 797
89 928
99 1067
```

Plot:



10.

a. Implement Warshall's Algorithm to find the transitive closure of a given directed graph.

Program:

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

int count = 0;
int max(int a,int b){
        return a>b?a:b;
}
void warshall(int A[100][100],int n){
        for(int i = 0;i<n;i++)
                for(int j = 0;j<n;j++)
                        for(int k = 0;k<n;k++){
                                count++;
                                A[j][k]=max(A[j][k],A[j][i]&&A[i][k]);
                        }
}
void analysis(){

        FILE *f;
        int AM[100][100];
        srand(time(NULL));
        for(int i = 4;i<=10;i++){
                f = fopen("data.txt","a");
                for(int j = 0;j<i;j++)
                        for(int k = 0;k<i;k++){
                                int ri = rand()%i;
                                int ci = rand()%i;
                                if(ri==ci)
                                        continue;
                                AM[ri][ci] = 1;
                        }
```

```
                count = 0;
                warshall(AM,i);
                fprintf(f,"%d %d\n",i,count);
                count =0 ;
                fclose(f);
        }

}

int main(){
        int ch;
        analysis();
}
```

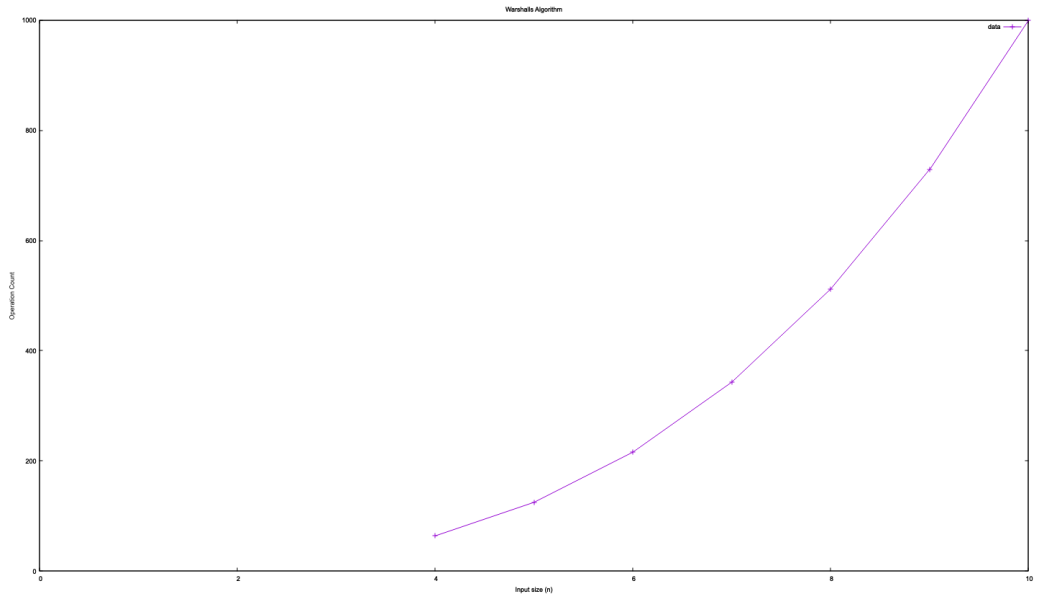
Output:

```
Enter the numbe of vertices
4
Enter the adjacency matrix
0 1 1 0
1 0 1 0
1 1 0 1
0 0 1 0
Transitive closure of the graph is:
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
```

Analysis:

```
4 64
5 125
6 216
7 343
8 512
9 729
10 1000
```

Plot:



b. Find All-pair shortest path for a given graph using Floyd's Algorithm.

Program:

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

int Min(int a,int b){
        return a<b?a:b;
}

void floyd(int *w[100],int v){

        for(int k=1;k<=v;k++)
                for(int i = 1;i<=v;i++)
                        for(int j =1;j<=v;j++)
                                w[i][j] = Min(w[i][j],w[i][k]+w[k][j]);

}

int main(){
        int v;
        printf("Enter the number of vertices\n");
                scanf("%d",&v);
```

```
        int *w[v+1];
        for(int i = 0;i<=v;i++)
                w[i] = (int *)malloc(sizeof(int)*(v+1));
        printf("Enter the weighted adjacency matrix(Enter 10000 for infinity)\n");
        for(int i = 1;i<=v;i++)
                for(int j = 1;j<=v;j++){
                        scanf("%d",&w[i][j]);
                }
        floyd(w,v);
        printf("All pairs shortest path: \n");
        for(int i =1;i<=v;i++){
                for(int j =1;j<=v;j++)
                        printf("%d ",w[i][j]);
                printf("\n");
        }
}
```

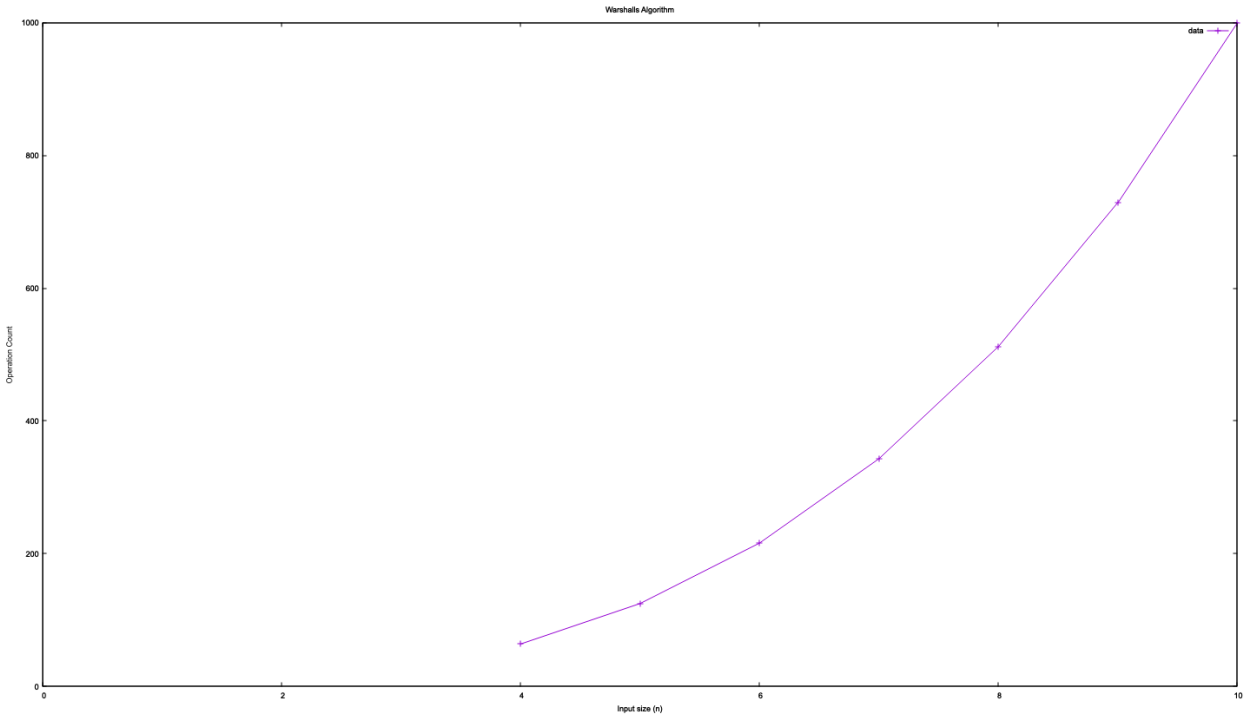
Output:

```
Enter the number of vertices
4
Enter the weighted adjacency matrix(Enter 10000 for infinity)
0 5 8 10000
9 0 19 10000
7 8 0 8
10000 10000 9 0
All pairs shortest path:
0 5 8 16
9 0 17 25
7 8 0 8
16 17 9 0
```

Analysis:

```
4 64
5 125
6 216
7 343
8 512
9 729
10 1000
```

Plot:



11.

a. Implement 0/1 Knapsack problem using Dynamic Programming.

1. **Without memory function:**
   Program:

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


typedef struct table{
        int item,value,weight;
}T;

int optimal,subset[20],count;

int max(int a,int b){
        return a>b?a:b;
}

void knapsack(T *t,int n,int W){
```

```
	int *a[n+1];
	for(int i = 0;i<=n;i++)
		a[i] = (int *)malloc(sizeof(int)*(W+1));

	for(int i = 0;i<=n;i++){
		//printf("HI\n");
		for(int j = 0;j<=W;j++){
			count++;
			if(i == 0||j == 0){
				a[i][j] = 0;
				//continue;
			}
			else if(j-t[i].weight<0)
				a[i][j] = a[i-1][j];
			else
				a[i][j] =
max(a[i-1][j],t[i].value+a[i-1][j-t[i].weight]);
		}
	}
	//printf("HI\n");
	optimal = a[n][W];
	int RC = W;
	int i = n;
	int j = 0;
	while(RC!=0 && i>=1){
		//printf("HI\n");
		if(a[i][RC]!=a[i-1][RC]){
			subset[j] = i;
			j++;
			RC = RC - t[i].weight;
		}
		i--;
	}
	//printf("HI\n");
}


int main(){
	int n;
	printf("Enter the number of items\n");
```

```
        scanf("%d",&n);
    T *t = (T *)malloc(sizeof(struct table)*(n+1));
    printf("Enter item,value and weight\n");
    for(int i = 1;i<=n;i++)
        scanf("%d%d%d",&t[i].item,&t[i].weight,&t[i].value);
    printf("Enter knapsack capacity\n");
    int W;
    scanf("%d",&W);
    knapsack(t,n,W);
    printf("Optimal value: %d\n",optimal);
    printf("Subset\n");
    for(int i = 0;i<n;i++)
        printf("%d, ",subset[i]);
    printf("\nCount: %d\n",count);
}
```

Output:

```
Enter the number of items
4
Enter item,value and weight
1 5 2
2 3 1
3 4 3
4 5 6
Enter knapsack capacity
7
Optimal value: 6
Subset
4, 0, 0, 0,
Count: 40
```

2. **With memory function:**
   Program:

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

typedef struct table{
    int value,item,weight;
}T;
```

```
T *t;
int *a[20],optimal,subset[10],count;

int max(int a,int b){
	return a>b?a:b;
}

int knapsack(int i,int j){
	if(a[i][j]==-1){
		count++;
		if(j-t[i].weight<0)
			a[i][j] = knapsack(i-1,j);
		else
			a[i][j] =
max(knapsack(i-1,j),knapsack(i-1,j-t[i].weight)+t[i].value);
	}
	return a[i][j];
}

int main(){
	int n;
	printf("Enter the number of items\n");
	scanf("%d",&n);

	t = (T *)malloc(sizeof(struct table)*(n+1));

	printf("Enter the item, weight and value\n");
	for(int i = 1;i<=n;i++)
		scanf("%d%d%d",&t[i].item,&t[i].weight,&t[i].value);
	printf("Enter the capacity\n");
	int W;
	scanf("%d",&W);
	for(int i = 0;i<=n;i++){
		int j;
		a[i] = (int *)malloc(sizeof(int)*(W+1));
		for(int j = 0;j<=W;j++){
			if(i==0||j==0)
			a[i][j] = 0;
		else
			a[i][j] = -1;
```

```
                }

                //printf("%d\n",a[i][j]);
        }
        optimal = knapsack(n,W);
        printf("Optimal value: %d\n",optimal);
        int RC = W;
        int i = n;
        int j = 0;
        while(RC!=0 && i>=1){
                if(a[i][RC]!=a[i-1][RC]){
                        subset[j] = i;
                        j++;
                        RC = RC-t[i].weight;
                }
                i--;
        }
        printf("subset\n");
        for(i = 0;i<j;i++)
                printf("%d, ",subset[i]);
        printf("\n");
        printf("Count = %d\n",count);
}
```

Output:

```
Enter the number of items
4
Enter the item, weight and value
1 2 5
2 1 3
3 3 4
4 6 5
Enter the capacity
8
Optimal value: 12
subset
3, 2, 1,
Count = 12
```

B. **Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.**

Program:

```
#include<stdio.h>
#include<stdlib.h>
int n;
int *a[30];

int prims(){
	int edges = 0,count = 0,cost = 0;
	int row,col;
	int selected[n];
	int min;
	for(int i = 0;i<n;i++)
		selected[i] = 0;
	selected[0] = 1;
	while(edges < n-1){
		row = 0;
		col = 0;
		min = 999;
		count++;
		for(int i = 0;i<n;i++){
			count++;
			if(selected[i]){
				//count++;
				for(int j = 0;j<n;j++){
					//count++;
					if(selected[j]==0 && a[i][j]){
						//count++;
						if(min > a[i][j]){
							//count++;
							min = a[i][j];
							row = i;
							col = j;
						}
					}
				}
			}
```

```
                }
                printf("V%d ---- %d ---- V%d\n",row+1,a[row][col],col+1);
                cost += a[row][col];
                selected[col] = 1;
                edges++;
        }
        printf("Weight of MST: %d\n",cost);
        return count;
}


int main(){
        printf("Enter the number of vertices\n");
                scanf("%d",&n);
        for(int i = 0;i<n;i++)
                a[i] = (int *)malloc(sizeof(int)*n);
        printf("Enter the adjacency matrix\n");
        for(int i = 0;i<n;i++)
                for(int j = 0;j<n;j++)
                        scanf("%d",&a[i][j]);
        printf("Count: %d\n",prims());
}
```

Output:

```
Enter the number of vertices
4
Enter the weighted adjacency matrix
0 5 6 0
8 0 9 0
4 5 0 9
0 0 8 0
V1 ---- 5 ---- V2
V1 ---- 6 ---- V3
V3 ---- 9 ---- V4
Weight pf MST is: 20

Count = 17
```