Knapsac Algorithm

```
#include<iostream>
using namespace std;

void knapsack(int n, float wt[], float pft[], float capy)
{
	float x[20], tp = 0;
	int i, j, u;
	u = capy;

	for (i = 0; i < n; i++)
	{
		x[i] = 0.0;
	}

	for (i = 0; i < n; i++)
	{
		if(wt[i] > u)
			break;
		else
		{
			x[i] = 1.0;
			tp = tp + pft[i];
			u = u - wt[i];
		}
	}

	if (i < n)
		x[i] = u / wt[i];
		tp = tp + (x[i] * pft[i]);
		cout << "The result vector is: " << endl;

	for(i = 0; i < n; i++)
		cout << x[i];
		cout << "Maximum profit is: " << tp << endl;
}

int main()
{
	float wt[20], pft[20], capy;
	int num, i, j;
	float ratio[20], temp;

	cout << "Enter the no. of objects: ";
	cin >> num;
	cout << "Enter the wts and profits of each object: " << endl;

	for (i = 0; i < num; i++)
	{
		cin >> wt[i] >> pft[i];
	}
```

```
        cout << "Enter the capacity of knapsack: ";
        cin >> capy;
        cout << endl;

        for(i = 0; i < num; i++)
        {
                ratio[i] = pft[i] / wt[i];
        }
        for (i = 0; i < num; i++)
        {
                for (j = i + 1; j < num; j++)
                {
                        if (ratio[i] < ratio[j])
                        {
                                temp = ratio[j];
                                ratio[j] = ratio[i];
                                ratio[i] = temp;

                                temp = wt[j];
                                wt[j] = wt[i];
                                wt[i] = temp;

                                temp = pft[j];
                                pft[j] = pft[i];
                                pft[i] = temp;
                        }
                }
        }

knapsack(num, wt, pft, capy);

return(0);
}

/*output
Enter the no. of objects: 5
Enter the wts and profits of each object:
6
3
6
4
1
5
2
21
5
4
Enter the capacity of knapsack: 15

The result vector is:
11110.166667Maximum profit is: 34.5    */
```

Quick Sort Algorithm

```
#include <iostream>
using namespace std;

void quick(int a[], int lw, int up);
int party(int arr[], int lw, int up);

int main ()
{
	int arr[] = {12, 85, 69, 74, 25, 65, 71, 692};
	int n = 8, i;

	cout << "The unsorted array is : ";
	for(i=0; i<n; i++)
	{
		cout << arr[i] << " ";
	}

	quick(arr, 0, n-1);

	cout << "The sorted array is : ";
	for(i=0; i<n; i++)
	{
		cout << arr[i] << " ";
	}

	cout << endl;
}

void quick(int arr[], int lw, int up)
{
	int pvtloc;
	if(lw >= up)
	return;

	pvtloc = party(arr, lw, up);
	quick(arr, lw, pvtloc-1);
	quick(arr, pvtloc+1, up);
}

int party(int arr[], int lw, int up)
{
	int temp, i, j, pivot;

	i = lw+1; j = up;
	pivot = arr[lw];

	while(i<=j)
	{
		while((arr[i] < pivot) && (i < up))
		{
```

```
                i++;
            }

            while(arr[j] > pivot)
            {
                j--;
            }

            if(i<j)
            {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;

                i++;
                j--;
            }
            else
            {
                i++;
            }


        arr[lw] = arr[j];
        arr[j] = pivot;

        return j;
}
}

/*output
The unsorted array is : 12 85 69 74 25 65 71 692
The sorted array is : 12 25 65 69 71 74 85 692
*/
```

Bellman Ford Algorithm

```
#include <iostream>
using namespace std;
#define MAX 100
#define infty 9999
#define NIL -1
#define TRUE 1
#define FALSE 0
int n, front, rear;
int adj[MAX][MAX];
int prede[MAX];
int pl[MAX];
int ispqueue[MAX], queue[MAX], dltqueue(), emptqueue();
void intlqueue();
void insrtqueue(int u);
void cretgraph();
void fndpath(int s, int v);
int bellman(int s);

int main()
{
	int flag, s, v;
	cretgraph();
	cout << "Enter source vertex : ";
	cin >> s;
	flag = bellman(s);

	if(flag == -1)
	{
		cout << "Error : Negative cycle in graph" << endl;
		exit(1);
	}
	while(1)
	{
		cout << "Enter destination vertex : ";
		cin >> v;

		if(v == -1)
			break;
		if(v<0 || v>=n)
			cout << "This vertex doesn't exists" << endl;
		else if(v==s)
			cout << "Source and destination vertex same" << endl;
		else if(pl[v] == infty)
			cout << "There is no path from source to destination" << endl;
		else
			fndpath(s, v);
	}
}

void fndpath(int s, int v)
```

```
{
        int i, u, path[MAX], shortdist = 0, count = 0;

        while(v != s)
        {
                count++;
                path[count]++;
                u = prede[v];
                shortdist += adj[u][v];
                v = u;
        }
        count++;
        path[count] = s;
        cout << "Shortest path is : ";
        for(i = count; i>=1; i--)
                cout << path[i] << endl;
        cout << "Shortest dist. is : " << shortdist;
}

int bellman(int s)
{

        int k=0, i, current;

        for(i=0; i<n; i++)
        {
                prede[i] = NIL;
                pl[i] = infty;
                ispqueue[i] = FALSE;
        }

        intlqueue();
        pl[s]= 0;
        insrtqueue(s);
        ispqueue[s] = TRUE;

        while(!emptqueue())
        {
                current = dltqueue();
                ispqueue[current] = FALSE;
                if(s==current)
                        k++;
                if(k>n)
                        return -1;
                for(i=0; i<n; i++)
                {
                        if(adj[current][i] != 0)
                                if (pl[i] > pl[current] + adj[current][i])
                                {
                                        pl[i] = pl[current] + adj[current][i];
                                        prede[i] = current;
                                        if(!ispqueue[i])
```

```
                                        {
                                                insrtqueue(i);
                                                ispqueue[i]=TRUE;
                                        }
                                }
                        }
                }
                return 1;
}

void intlqueue()
{
        for(int i=0; i<MAX; i++)
                queue[i] = 0;
        rear = -1; front = -1;
}

int emptqueue()
{
        if(front==-1 || front>rear)
                return 1;
        else
                return 0;
}

void insrtqueue(int added_item)
{
        if (rear==MAX-1)
        {
                printf("Queue Overflow\n");
                exit(1);
        }
        else
        {
                if (front==-1)
                        front = 0;
                rear = rear+1;
                queue [rear] = added_item;
        }
}

int dltqueue()
{
        int d;
        if(front==-1 || front>rear)
        {
                printf("Queue Underflow\n");
                exit(1);
        }
        else
        {
                d = queue [front];
```

```
                front = front+1;
        }
        return d;
}

void cretgraph()
{
        cout<<"enter the number of vertice:";
        cin>>n;

        cout<<"enter the adjacency matrix:\n";
        for(int i = 0;i<n;i++)
        {
                for (int j = 0; j<n;j++)
                {
                        cin >>adj[i][j];
                }
        }
}

/* output
enter the number of vertice:4
enter the adjacency matrix:
1
2
3
4
1
3
2
4
1
4
2
3
4
2
3
2
Enter source vertex : 2
Enter destination vertex : 4
This vertex doesn't exists
Enter destination vertex : 3
Shortest path is : 2
1
Shortest dist. is : 3Enter destination vertex :
*/
```

Merge Sort Algorithm

```
#include<iostream>
using namespace std;

void merges(int a[], int l, int up);
void merge(int arr[], int temp[], int l1, int up1, int l2, int up2);
void copy(int arr[], int temp[],int low, int up);
int main()
{
	int a[7]={2,33,54,6,87,11,12},i;
	int n=7;
	merges(a,0,n-1);

	cout << "The unsorted array is : ";
	for(i=0; i<n; i++)
	{
		cout << a[i] << " ";
	}

	cout << "\nThe sorted array is : ";
	for(i=0; i<n; i++)
	{
		cout<<a[i];
	}
	return 0;
}

void merges(int a[], int l, int up)
{
	int mid;
	int temp[10];
	if(l<up)
	{
	mid=(l+up)/2;
	merges(a,l,mid);
	merges(a,mid+1,up);
	merge(a,temp,l,mid,mid+1,up);
	copy(a,temp,l,up);
	}
}

void merge(int arr[],int temp[], int l1, int up1, int l2, int up2)
{
	int i=l1;
	int j=l2;
	int k=l1;

	while((i<=up1)&&(j<=up2))
	{
		if(arr[i]<arr[j])
			temp[k++]=arr[i++];
```

```
                else
                        temp[k++]=arr[j++];
        }
        while(i<=up1)
                temp[k++]=arr[i++];
        while(j<=up2)
                temp[k++]=arr[j++];
}
void copy(int arr[], int temp[], int low, int up)
{
        int i;
        for(i=low; i<=up; i++)
        {
                arr[i]=temp[i];
        }
}

/*output
The unsorted array is : 2 6 11 12 33 54 87
The sorted array is : 261112335487
*/
```