GREEDY ALGORITHMS

1. ACTIVITY SELECTION PROBLEM

Time: O(nlogn) Aux. Space:O(1)

```
bool myCmp(pair <int, int> a, pair <int, int> b)
{
        return (a.second < b.second);
}

int maxActivities(pair <int, int> arr[], int n)
{
        sort(arr, arr + n, myCmp);

        int prev = 0;
        int res = 1;

        for(int curr = 1; curr < n; curr++)
        {

                if(arr[curr].first >= arr[prev].second)
                {
                        res++;

                        prev = curr;
                }
        }

        return res;
}
```

2. FRACTIONAL KNAPSACK

Time: O(N log N) Aux. Space:O(1)

```
bool myCmp(pair <int, int> a, pair <int, int> b)
{
        double r1 = (double)a.first / a.second;

        double r2 = (double)b.first / b.second;

        return r1 > r2;
}

double fKnapS(int W, pair <int, int> arr[], int n)
{
        sort(arr, arr + n, myCmp);

        double  res = 0.0;

        for(int i = 0; i < n; i++)
        {
                if(arr[i].second <= W)
                {
                        res += arr[i].first;

                        W = W - arr[i].second;
                }
                else
                {
                        res += arr[i].first * ((double) W / arr[i].second);

                        break;
                }
        }
```

```
        return res;
}
```

3. JOB SCHEDULING

Time: O(n2)

```
#include <iostream>
#include <algorithm>
using namespace std;
struct Job
{
   char id;     // Job Id
   int dead;    // Deadline of job
   int profit;  // Profit if job is over before or on deadline
};
bool comparison(Job a, Job b)
{
    return a.profit>b.profit;
}
void printJobScheduling(Job arr[],int n)
{
    sort(arr,arr+n,comparison);
    int result=0;
    bool slot[n];
    for(int i=0;i<n;i++)
    {
        slot[i]=false;
    }
    for(int i=0;i<n;i++)
    {
        for(int j=min(n,arr[i].dead)-1;j>=0;j--)
        {
            if(slot[j]==false)
```

```
            {
                result+=arr[i].profit;
                slot[j]=true;
                break;
            }
        }
    }
    cout<<result;
}
int main()
{
    Job arr[] = { {'a', 2, 100}, {'b', 1, 19}, {'c', 2, 27},
                 {'d', 1, 25}, {'e', 3, 15}};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Following is maximum profit sequence of jobs \n";

    // Function call
    printJobScheduling(arr, n);
    return 0;
}
```

4. HUFFMAN CODING

Time:O(nlogn) Aux. Space: O(n)

```
#include <bits/stdc++.h>
using namespace std;
struct Node {
        char data;
        unsigned freq;
        Node *left, *right;
        Node(char data, unsigned freq, Node* l = NULL, Node* r =
NULL)
        {
                this->left = l;
```

```
		this->right = r;
		this->data = data;
		this->freq = freq;
	}
};
struct compare {
	bool operator()(Node* l, Node* r)
	{
		return (l->freq > r->freq);
	}
};
void printCodes(struct Node* root, string str)
{
	if (!root)
		return;
	if (root->data != '$')
		cout << root->data << ": " << str << "\n";
	printCodes(root->left, str + "0");
	printCodes(root->right, str + "1");
}
void printHcodes(char arr[], int freq[], int size)
{
	priority_queue<Node*, vector<Node*>, compare> h;
	for (int i = 0; i < size; ++i)
		h.push(new Node(arr[i], freq[i]));
	while (h.size() > 1) {
		Node *l = h.top();h.pop();
		Node *r = h.top();h.pop();
		Node *top = new Node('$', l->freq + r->freq, l, r);
		h.push(top);
	}
	printCodes(h.top(), "");
}
int main()
{
```

```
        char arr[] = { 'a', 'd', 'e', 'f' };
        int freq[] = { 30, 40, 80, 60 };
        int size = sizeof(arr) / sizeof(arr[0]);
        printHcodes(arr, freq, size);

        return 0;
}
```