NAME:- Sourav Paul
GROUP:- D2
REG. NO:- 20214056
BRANCH:- CSE DEPT.

ASSIGNMENT - 06

Q1. Given an array A of size N and a number K(where k<N). Find the K-th largest/smallest number in the array, i.e., K-th order statistic.

Ans:-

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

int partition(int arr[], int l, int r);

int kthSmallest(int arr[], int l, int r, int K)
{
        if (K > 0 && K <= r - l + 1) {
                int pos = partition(arr, l, r);
                if (pos - l == K - 1)
                        return arr[pos];
                if (pos - l > K - 1)
                        return kthSmallest(arr, l, pos - 1, K);
                return kthSmallest(arr, pos + 1, r,
                                                K - pos + l - 1);
        }
        return INT_MAX;
}
```

```
void swap(int* a, int* b)
{
	int temp = *a;
	*a = *b;
	*b = temp;
}

int partition(int arr[], int l, int r)
{
	int x = arr[r], i = l, j;
	for (j = l; j <= r - 1; j++) {
		if (arr[j] <= x) {
			swap(&arr[i], &arr[j]);
			i++;
		}
	}
	swap(&arr[i], &arr[r]);
	return i;
}

int main()
{
	int n, i, k;
	printf("Enter array size : ");
	scanf("%d", &n);
	printf("Enter k : ");
	scanf("%d", &k);

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

	printf("K'th smallest element is %d", kthSmallest(arr, 0, n - 1, k));
```

```
        return 0;
}
/*OUTPUT
Enter array size : 6
Enter k : 3
Enter array : 7 10 4 3 20 15
K'th smallest element is 7%
*/
```

Q2)You are given n activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.

```
Ans:-  #include <stdio.h>

void printMaxActivities(int s[], int f[], int n)

{

        int i, j;

        printf("Following activities can be performed : ");

        i = 0;

        printf("%d ", i);


        for (j = 1; j < n; j++) {

                if (s[j] >= f[i]) {

                        printf("%d ", j);
                i = j;
            }
        }
}

int main()
{
        int n, i;
        printf("Enter number of activities : ");
        scanf("%d", &n);
        int s[n];
        int f[n];

        printf("Enter start time : ");
        for(i=0; i<n; i++)
                scanf("%d", &s[i]);
        printf("Enter end time : ");
        for(i=0; i<n; i++)
                scanf("%d", &f[i]);

        printMaxActivities(s, f, n);
        printf("\n");
```

```
    return 0;
}

/*OUTPUT
Enter number of activities : 6
Enter start time : 1 3 0 5 8 5
Enter end time : 2 4 6 7 9 9
Following activities can be performed : 0 1 3 4
*/
```

Q3) We have some coin denominations say (1,5,10,20,50), make the change for amount S using the smallest number of coins possible.

```
Ans:-  #include <stdio.h>
int coinCount[5] = {0};
int coins[] = {1, 5, 10, 20, 50};

int count(int n, int sum)
{
    if (sum == 0)
        return 1;
    if (sum < 0)
```

```
        return 0;
    if (n <= 0)
        return 0;

    coinCount[n-1]=(sum/coins[n-1]);
    count(n-1, sum%coins[n-1]);
}

int main()
{
    int i, j, money;
    int n = sizeof(coins) / sizeof(coins[0]);
    printf("Enter money to be changed : ");
    scanf("%d", &money);

    count(n, money);
    for(i=4; i>=0; i--)
        printf("Rs %d x %d\n", coins[i], coinCount[i]);
    return 0;
}
/*OUTPUT
Enter money to be changed : 37
```

```
Rs 50 x 0

Rs 20 x 1

Rs 10 x 1

Rs 5 x 1

Rs 1 x 2

*/
```

Q4) Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack.

```
Ans:-  #include<iostream>

using namespace std;


struct Item {

   int value, weight;


   Item(int value, int weight)

   {

      this->value = value;

      this->weight = weight;

   }

};
```

```
static bool cmp(struct Item a, struct Item b)
{
    double r1 = (double)a.value / (double)a.weight;
    double r2 = (double)b.value / (double)b.weight;
    return r1 > r2;
}

double fractionalKnapsack(int W, struct Item arr[], int N)
{
    sort(arr, arr + N, cmp);

    double finalvalue = 0.0;

    for (int i = 0; i < N; i++) {

        if (arr[i].weight <= W) {
            W -= arr[i].weight;
            finalvalue += arr[i].value;
        }

        else {
}
            finalvalue += arr[i].value * ((double)W /
(double)arr[i].weight);
            break;
        }
    }

    return finalvalue;
}



int main()
{
    int W = 50;
    Item arr[] = { { 60, 10 }, { 100, 20 }, { 120, 30 } };

    int N = sizeof(arr) / sizeof(arr[0]);

    cout << " Maximum value we can obtain " << "  " <<
fractionalKnapsack(W, arr, N) << endl;
    return 0;
}

/*OUTPUT
```

```
 Maximum value we can obtain  240
 */
```

Q5) Write a c program to implement huffman coding using greedy algorithm.

Ans:-

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

// A Huffman tree node
struct MinHeapNode {

        char data;

        unsigned freq;

        MinHeapNode *left, *right;

        MinHeapNode(char data, unsigned freq)

        {
                left = right = NULL;

                this->data = data;

                this->freq = freq;

        }

};


struct compare {


        bool operator()(MinHeapNode* l, MinHeapNode* r)


        {

                return (l->freq > r->freq);

        }

};


void printCodes(struct MinHeapNode* 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 HuffmanCodes(char data[], int freq[], int size)
{
    struct MinHeapNode *left, *right, *top;

    // Create a min heap & inserts all characters of data[]
    priority_queue<MinHeapNode*,
vector<MinHeapNode*>,compare> minHeap;

    for (int i = 0; i < size; ++i)
        minHeap.push(new MinHeapNode(data[i], freq[i]));

    while (minHeap.size() != 1) {

        left = minHeap.top();
        minHeap.pop();
        right = minHeap.top();
        minHeap.pop();

        top = new MinHeapNode('$', left->freq + right->freq);

        top->left = left;
        top->right = right;

        minHeap.push(top);
    }

    printCodes(minHeap.top(), " ");
}


int main()
{

    char arr[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
    int freq[] = { 5, 9, 12, 13, 16, 45 };
    int size = sizeof(arr) / sizeof(arr[0]);

    HuffmanCodes(arr, freq, size);

    return 0;
}

/*OUTPUT
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111
*/
```