



**Data Structures Applications Laboratory (21EECF201/23EVTF203)**

**Name of the student: Vinay Matade**

**USN: 01FE23BEC239**

**Div: E**

**Roll No: 508**

**Task 1:** Write a program to generate N random numbers between a given range of numbers (P, Q) and write it to a file (*Input.txt*).

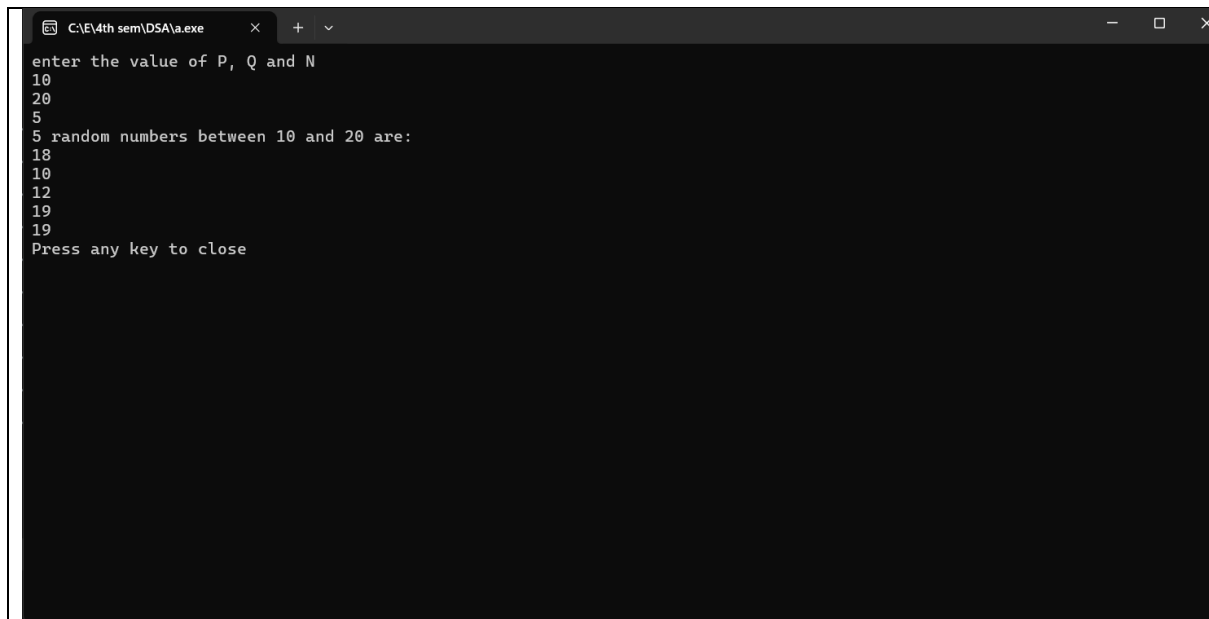
Code:

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

int main()
{
    FILE *inputFile;
    inputFile=fopen("input.txt", "w");
    if (inputFile == NULL)
    {
        printf("File not found\n");
        return 1;
    }
    int r,p,q,n;
    printf("enter the value of P, Q and N\n");
    scanf("%d%d%d",&p,&q,&n);
    srand(time(0));
    printf("%d random numbers between %d and %d are:\n",n,p,q);
    for(int i=0;i<n;i++)
    {
        r= p+rand()%(q-p+1);
        printf("%d\n",r);
        fprintf(inputFile, "%d ", r);
    }
    fclose(inputFile);
    printf("Press any key to close\n");
    char a;
    scanf(" %c",&a);
    return 0;
}
```

Output: Put screenshots of the output

## **Data Structures Applications Laboratory (21EECF201/23EVTF203)**



```
C:\E\4th sem\DSA\1a.exe
enter the value of P, Q and N
10
20
5
5 random numbers between 10 and 20 are:
18
10
12
19
19
Press any key to close
```

**Task 2:** List any 10 sorting algorithms. Describe the operation of each sorting algorithm in few sentences

1. Quick sort: Divide and conquer algorithm. Chose a pivot number in an unsorted array which can be the first, middle or last number. Pivot is then used to compare all values into one of each list, less than or equal to the pivot or more than the pivot. A pivot number is then chosen from the newly created array and does another comparison that is greater or smaller. It is done repeatedly until the array is sorted. Worst case:  $O(N^2)$ , Best Case:  $O(N \log N)$
2. Bubble sort: Goes through the entire array. Compares the number with the next number. If the second number is larger, swaps the numbers and moves to the next pair of numbers until all the numbers are sorted. Maximum number of operations is  $N^2$ , minimum is  $N$ .
3. Merge Sort: Divide and conquer algorithm. Divide the array into smallest possible elements. Compare the element with the element next to it, sort and merge the array. Continue until you have a single array that is sorted. Time taken:  $O(N \log N)$
4. Selection sort: Goes through the array and finds the smallest number. Puts the smallest item first and again checks for the next smallest number, puts it in the second place and repeats the process until the array is sorted. Maximum and minimum operations are  $N^2$ .
5. Insertion Sort: Creates 2 arrays, one sorted and other unsorted. It then iterates all the values and place the number from unsorted array into the

**Data Structures Applications Laboratory (21EECF201/23EVTf203)**

sorted array where it belongs to by iterating constantly and check if the number should be between them. Worst case:  $O(N^2)$ , best case:  $O(N)$

6. Heap Sort: Comparison based algorithm. Take the unsorted array and create a tree like structure. Once tree is built, top element switches with the element at the end, heap is then rebuilt and number switch again until it is sorted. Time taken:  $O(N \log N)$ .

7. Counting Sort: 3 arrays required, unsorted, counting and output array. Find the maximum magnitude element in the array which is used to define the size of the counting array. If the array's position number exists in the unsorted array, put 1 in the counting array. Then add the numbers in the counting array sequentially which tells us the number of elements before it. Go in reverse order of the input array and insert it in the index that is specified by the value of the number. Subtract by 1, then repeat until everything is sorted. Time taken:  $O(N+K)$ ,  $K$  = Range of smallest and largest number.

8. Shell Sort: In place comparison algorithm. It uses intervals, i.e., length of the array / 2, then compare the last element of each of the interval (2 newly created arrays). If first number is larger than second number, then swap. Divide the interval into (the size of previously divided array - 1) again and repeat until interval hits 1. Then use insertion sort to sort the array. Worst case:  $O(N^2)$ , Best case:  $O(N \log N)$ .

9. Tim Sort: Divide the array into smaller arrays called runs. These runs are then sorted using insertion sort, once the created runs are sorted, use merge sort on 2 smallest arrays, and then take the next run and sort it into the first array until everything is sorted. Time taken:  $O(N \log N)$ .

10. Radix Sort: Check the units digit of each element and sort it by that number. Then check the digit before the unit digit and uses that to sort. If the digit isn't available, then it is taken to be 0. Keep doing this again and again until sorted. Time taken =  $O(d*(N+b))$ , where  $d$  = Number of digits in the largest number,  $b$  = base of the numbers ( $10 = 0$  to  $9$ ).

**Task 3:** Use the file *Input.txt* (output of Task1) and implement each of the sorting algorithms (implement all 10 algorithms) – Each of the sorting algorithm can be in a different table

Code:

BucketSort:

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

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

```
void readFile(int arr[],int n, char *filename);
```

```
int bucketSort(int arr[], int n);
```

**Data Structures Applications Laboratory (21EECF201/23EVTF203)**

```
int main()
{
    int n;
    printf("Enter the number of elements in the array\n");
    scanf("%d",&n);
    int arr[n];
    readFile(arr,n,"input.txt");
    printf("array before sorting:\n");

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

    printf("\n");

    bucketSort(arr,n);
    printf("array after sorting:\n");

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

    printf("\n");
    return 0;
}

void readFile(int arr[],int n,char *filename)
{
    FILE *input;
    input=fopen(filename,"r");
    if (input == NULL)
    {
        printf("File couldn't be found\n");
        exit(1);
    }
    else
    {
        for(int i=0;i<n;i++)
            fscanf(input, "%d", &arr[i]);

        fclose(input);
    }
}

int bucketSort(int arr[], int n)
```

**Data Structures Applications Laboratory (21EECF201/23EVTF203)**

```
{
    int i,j;
    int max=arr[0];
    int is_sorted=1;
    for(i=0;i<n-1;i++)
    {
        if(arr[i]<arr[i-1])
        {
            is_sorted=0;
            break;
        }
    }

    for(i=1;i<n;i++)
    {
        if (arr[i]>max)
            max=arr[i];
    }
    int bucket[max+1];
    for (i=0;i<=max;i++)
        bucket[i] = 0;

    for (i=0;i<n;i++)
        bucket[arr[i]]++;

    for(i=0,j=0;i<=max;i++)
    {
        while(bucket[i]>0)
        {
            arr[j++]=i;
            bucket[i]--;
        }
    }
}
```

BubbleSort:

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

```
void loadArrayFromFile(int arr[], int n, char *filename);
void bubbleSort(int arr[], int n);
```

```
int main()
```

**Data Structures Applications Laboratory (21EECF201/23EVTf203)**

```
{
    int n=10;
    int arr[n];

    loadArrayFromFile(arr,n,"input.txt");

    printf("Before sorting:\n");
    for (int i=0;i<n;i++)
        printf("%d ",arr[i]);
    printf("\n");

    printf("Sorting in progress...\n");

    bubbleSort(arr, n);

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

    return 0;
}

void loadArrayFromFile(int arr[], int n, char *filename) {
    FILE *input = fopen(filename, "r");
    if (input == NULL) {
        printf("Error: Could not open file %s\n", filename);
        exit(1);
    }

    printf("Loading numbers from file...\n"); //debug print

    for (int i = 0; i < n; i++)
    {
        if (fscanf(input, "%d", &arr[i]) != 1)
        { // Handle read errors
            printf("Error reading number at index %d\n", i);
            fclose(input);
            exit(1);
        }
    }

    fclose(input);
```

**Data Structures Applications Laboratory (21EECF201/23EVTf203)**

```
}

void bubbleSort(int arr[], int n) {
    int swapped;

    for (int i = 0; i < n - 1; i++) {
        swapped = 0; // Assume no swaps
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap adjacent elements if out of order
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = 1; // Mark that a swap occurred
            }
        }
    }

    // Debugging(commented out)
    // printf("After pass %d: ",i+1);
    // for(int k=0;k<n;k++)
    // printf("%d ", arr[k]);
    // printf("\n");

    if (!swapped) {
        // Exit early if no swaps were made
        break;
    }
}
}
```

Counting Sort:

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

```
void readFile(int a[],int n,char *f);
void countingSort(int a[],int n);
```

```
int main() {
    int n=10,a[n];
    readFile(a,n,"input.txt");

    printf("Before:\n");
    for(int i=0;i<n;i++) printf("%d ",a[i]);
```

**Data Structures Applications Laboratory (21EECF201/23EVTF203)**

```
countingSort(a,n);

printf("\nAfter:\n");
for(int i=0;i<n;i++) printf("%d ",a[i]);

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

    printf("\n");
return 0;
}

void readFile(int a[],int n,char *f)
{
    FILE *in=fopen(f,"r");
    if(!in) exit(1);

    for(int i=0;i<n;i++) fscanf(in,"%d",&a[i]);
    fclose(in);
}

void countingSort(int a[],int n)
{
    int sorted=1;
    for(int i=1;i<n;i++)if(a[i]<a[i-1]) { sorted=0; break; }
    if(sorted) return;

    int max=a[0];
    for(int i=1;i<n;i++) if(a[i]>max) max=a[i];

    int *cnt=calloc(max+1,sizeof(int));
    for(int i=0;i<n;i++) cnt[a[i]]++;

    int idx=0;
    for(int i=0;i<=max;i++)
        while(cnt[i]--) a[idx++]=i;

    free(cnt);
}
```

HeapSort:



**Data Structures Applications Laboratory (21EECF201/23EVTF203)**

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

void loadNumbersFromFile(int arr[], int n, char *filename);
void heapSort(int arr[], int n);
void heapify(int arr[], int n, int i);

int main()
{
    int n=10; // Adjust for larger values of arrays
    int arr[n];

    loadNumbersFromFile(arr,n,"input.txt");

    printf("Before sorting:\n");
    for (int i=0;i<n;i++)
        printf("%d ",arr[i]);
    printf("\n");

    heapSort(arr,n);

    printf("Sorted:\n");
    for (int i=0;i<n;i++)
        printf("%d ",arr[i]);
    printf("\n");

    return 0;
}

void loadNumbersFromFile(int arr[], int n, char *filename)
{
    FILE *input = fopen(filename, "r");
    if (input == NULL)
    {
        printf("Error: Could not open file %s\n", filename);
        exit(1);
    }

    printf("Reading numbers from file...\n"); // A debug line

    for (int i = 0; i < n; i++) {
        if (fscanf(input, "%d", &arr[i]) != 1)
            { //file reading errors
```

**Data Structures Applications Laboratory (21EECF201/23EVTf203)**

```
        printf("Error reading number at index %d\n", i);
        fclose(input);
        exit(1);
    }
}

fclose(input);
}

void heapify(int arr[], int n, int i) {
    int largest = i;
    int leftChild = 2 * i + 1;
    int rightChild = 2 * i + 2;

    if (leftChild < n && arr[leftChild] > arr[largest])
        largest = leftChild;

    if (rightChild < n && arr[rightChild] > arr[largest])
        largest = rightChild;

    if (largest != i) {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;
        heapify(arr, n, largest);
    }
}

void heapSort(int arr[], int n) {
    // Checking if the array is already sorted
    int isSorted = 1;
    for (int i = 1; i < n; i++)
    {
        if (arr[i] < arr[i - 1])
        {
            isSorted = 0;
            break;
        }
    }
    if (isSorted)
    {
        printf("Array is already sorted, skipping heap sort.\n");
        return;
    }
}
```

**Data Structures Applications Laboratory (21EECF201/23EVTF203)**

```
}

// Build max heap
for (int i=n/2-1;i>=0;i--)
    heapify(arr, n, i);

// Extract elements from heap
for (int i = n - 1; i > 0; i--) {
    int temp = arr[0];
    arr[0] = arr[i];
    arr[i] = temp;
    heapify(arr, i, 0);
}
}

InsertionSort:
#include <stdio.h>
#include <stdlib.h>

void readFile(int ar[],int n,char *filename);
void insertionSort(int ar[],int n);

int main()
{
    int n = 10;
    int ar[n];
    readFile(ar, n, "input.txt");
    printf("My list before:\n");

    for (int i=0;i<n;i++)
        printf("%d ", ar[i]);
    printf("\n");

    insertionSort(ar, n);
    printf("My list after:\n");

    for (int i=0;i<n;i++)
        printf("%d ", ar[i]);

    printf("\n");
    return 0;
}
```

**Data Structures Applications Laboratory (21EECF201/23EVTf203)**

```
void readFile(int ar[], int n, char *filename)
{
    FILE *input;
    input=fopen(filename, "r");
    if (input==NULL)
    {
        printf("File couldn't be found\n");
        exit(1);
    }
    else
    {
        for(int i=0;i<n;i++)
            fscanf(input,"%d",&ar[i]);

        fclose(input);
    }
}
```

```
void insertionSort(int ar[], int n)
{
    int is_sorted=1;
    for (int i=1;i<n;i++)
    {
        if (ar[i]<ar[i-1])
        {
            is_sorted=0;
            break;
        }
    }
    if (is_sorted)
        return;
    for (int i =1;i<n;i++)
    {
        int a=ar[i];
        int j=i-1;
        while (j>=0 && ar[j]>a)
        {
            ar[j+1]=ar[j];
            j=j-1;
        }
        ar[j+1]=a;
    }
}
```

**Data Structures Applications Laboratory (21EECF201/23EVTf203)**

Mergesort:

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

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

```
void loadArrayFromFile(int arr[], int n, char *filename);
```

```
void mergeSort(int arr[], int l, int r);
```

```
void merge(int arr[], int l, int m, int r);
```

```
int main() {
```

```
    int n = 10;
```

```
    int arr[n];
```

```
    loadArrayFromFile(arr,n,"input.txt");
```

```
    printf("before sorting:\n");
```

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

```
        printf("%d ", arr[i]);
```

```
    printf("\n");
```

```
    mergeSort(arr,0,n-1);
```

```
    printf("after sorting:\n");
```

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

```
        printf("%d",arr[i]);
```

```
    printf("\n");
```

```
    return 0;
```

```
}
```

```
void loadArrayFromFile(int arr[], int n, char *filename)
```

```
{
```

```
    FILE *file = fopen(filename, "r");
```

```
    if (file == NULL)
```

```
    {
```

```
        printf("Error: Could not open file %s\n", filename);
```

```
        exit(1);
```

```
    }
```

```
    printf("Reading numbers from file...\n"); // Debug message
```

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

```
    {
```

**Data Structures Applications Laboratory (21EECF201/23EVTF203)**

```
        if (fscanf(file, "%d", &arr[i])!=1)
        { // Handle file read errors
            printf("Error: Failed to read number at index %d\n", i);
            fclose(file);
            exit(1);
        }
    }

    fclose(file);
}

void mergeSort(int arr[], int l, int r)
{
    if (l < r) {
        int m = l + (r - l) / 2; // Midpoint calculation

        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

void merge(int arr[], int l, int m, int r)
{
    int n1 = m - l + 1;
    int n2 = r - m;

    int leftArr[n1], rightArr[n2];

    for (int i = 0; i < n1; i++)
        leftArr[i] = arr[l + i];
    for (int j = 0; j < n2; j++)
        rightArr[j] = arr[m + 1 + j];

    int i = 0, j = 0, k = l;

    // Merge the two halves back into arr[]
    while (i < n1 && j < n2) {
        if (leftArr[i] <= rightArr[j]) {
            arr[k] = leftArr[i];
            i++;
        } else {
            arr[k] = rightArr[j];
            j++;
        }
        k++;
    }
    while (i < n1)
        arr[k++] = leftArr[i++];
    while (j < n2)
        arr[k++] = rightArr[j++];
}
```

**Data Structures Applications Laboratory (21EECF201/23EVTF203)**

```
        j++;
    }
    k++;
}

// Copy remaining elements from leftArr[]
while (i < n1)
{
    arr[k] = leftArr[i];
    i++;
    k++;
}

// Copy remaining elements from rightArr[]
while (j < n2)
{
    arr[k] = rightArr[j];
    j++;
    k++;
}
}

Quicksort:
#include <stdio.h>
#include <stdlib.h>

void loadArrayFromFile(int arr[], int n, char *filename);
void quicksort(int arr[], int low, int high);
int partition(int arr[], int low, int high);

int main()
{
    int n = 10;
    int arr[n];

    loadArrayFromFile(arr, n, "input.txt");

    printf("Array before sorting:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");

    quicksort(arr, 0, n - 1);
```

**Data Structures Applications Laboratory (21EECF201/23EVTF203)**

```
printf("Array after sorting:\n");
for (int i = 0; i < n; i++)
    printf("%d ", arr[i]);
printf("\n");

return 0;
}

void loadArrayFromFile(int arr[], int n, char *filename)
{
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        printf("Error: Could not open file %s\n", filename);
        exit(1);
    }

    printf("Reading numbers from file...\n"); // Debugging print

    for (int i = 0; i < n; i++) {
        if (fscanf(file, "%d", &arr[i]) != 1) { // Handle file read errors
            printf("Error: Failed to read number at index %d\n", i);
            fclose(file);
            exit(1);
        }
    }

    fclose(file);
}

void quicksort(int arr[], int low, int high)
{
    if (low < high)
    {
        // Partition the array, and get the pivot index
        int pivotIndex = partition(arr, low, high);

        // Recursively sort left and right subarrays
        quicksort(arr, low, pivotIndex - 1);
        quicksort(arr, pivotIndex + 1, high);
    }
}
```



**Data Structures Applications Laboratory (21EECF201/23EVTF203)**

```
int partition(int arr[], int low, int high)
{
    int pivot = arr[high]; // Choosing the last element as pivot
    int i = low - 1;       // i will track the position for swapping

    for (int j = low; j < high; j++)
    {
        if (arr[j] < pivot)
        {
            i++; // Move the smaller element to the correct position
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    // Swap pivot into its correct position
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;

    return i + 1; // Return the pivot index
}
```

Radixsort:

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

```
void loadArrayFromFile(int arr[], int n, char *filename);
void radixSort(int arr[], int n);
int getMax(int arr[], int n);
void countSort(int arr[], int n, int digitPlace);
```

```
int main()
{
    int n = 10;
    int arr[n];

    loadArrayFromFile(arr, n, "input.txt");

    printf("Array before sorting:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
}
```

**Data Structures Applications Laboratory (21EECF201/23EVTF203)**

```
printf("\n");

radixSort(arr, n);

printf("Array after sorting:\n");
for (int i = 0; i < n; i++)
    printf("%d ", arr[i]);
printf("\n");

return 0;
}

void loadArrayFromFile(int arr[], int n, char *filename)
{
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        printf("Error: Could not open file %s\n", filename);
        exit(1);
    }

    printf("Reading numbers from file...\n"); // Debugging print

    for (int i = 0; i < n; i++) {
        if (fscanf(file, "%d", &arr[i]) != 1) { // Handle file read errors
            printf("Error: Failed to read number at index %d\n", i);
            fclose(file);
            exit(1);
        }
    }

    fclose(file);
}

void radixSort(int arr[], int n)
{
    int maxVal = getMax(arr, n);

    // Process each digit's place value using counting sort
    for (int digitPlace = 1; maxVal / digitPlace > 0; digitPlace *= 10) {
        countSort(arr, n, digitPlace);
    }
}
```

**Data Structures Applications Laboratory (21EECF201/23EVTf203)**

```
int getMax(int arr[], int n) {
    int maxVal = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > maxVal)
            maxVal = arr[i];
    }
    return maxVal;
}

void countSort(int arr[], int n, int digitPlace) {
    int output[n];
    int count[10] = {0}; // Count occurrences of digits (0-9)

    // Count occurrences of each digit at the current place value
    for (int i = 0; i < n; i++)
        count[(arr[i] / digitPlace) % 10]++;

    // Convert count[] into a cumulative frequency array
    for (int i = 1; i < 10; i++)
        count[i] += count[i - 1];

    // Build the output array using the count array
    for (int i = n - 1; i >= 0; i--) {
        output[count[(arr[i] / digitPlace) % 10] - 1] = arr[i];
        count[(arr[i] / digitPlace) % 10]--;
    }

    // Copy sorted elements back into the original array
    for (int i = 0; i < n; i++)
        arr[i] = output[i];
}

Randomnumbers:
//N random numbers between (P,Q)
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
    FILE *inputFile;
    inputFile=fopen("input.txt", "w");
    if (inputFile == NULL)
```

**Data Structures Applications Laboratory (21EECF201/23EVTf203)**

```
{
    printf("File not found\n");
    return 1;
}
int r,p,q,n;
printf("enter the value of P, Q and N\n");
scanf("%d%d%d",&p,&q,&n);
srand(time(0));
printf("%d random numbers between %d and %d are:\n",n,p,q);
for(int i=0;i<n;i++)
{
    r= p+rand()%(q-p+1);
    printf("%d\n",r);
    fprintf(inputFile, "%d ", r);
}
fclose(inputFile);
printf("Press any key to close\n");
char a;
scanf(" %c",&a);
return 0;
}
```

Selectionsort:

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

```
void loadArrayFromFile(int arr[], int n, char *filename);
void selectionSort(int arr[], int n);
```

```
int main() {
    int n = 10;
    int arr[n];

    loadArrayFromFile(arr, n, "input.txt");

    printf("Array before sorting:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");

    selectionSort(arr, n);

    printf("Array after sorting:\n");
```

**Data Structures Applications Laboratory (21EECF201/23EVTf203)**

```
for (int i = 0; i < n; i++)
    printf("%d ", arr[i]);
printf("\n");

return 0;
}

void loadArrayFromFile(int arr[], int n, char *filename)
{
    FILE *file = fopen(filename, "r");
    if (file == NULL)
    {
        printf("Error: Could not open file %s\n", filename);
        exit(1);
    }

    printf("Reading numbers from file...\n"); // Debugging print

    for (int i = 0; i < n; i++)
    {
        if (fscanf(file, "%d", &arr[i]) != 1)
        { // Handle file read errors
            printf("Error: Failed to read number at index %d\n", i);
            fclose(file);
            exit(1);
        }
    }

    fclose(file);
}

void selectionSort(int arr[], int n)
{
    int isSorted = 1; // Assume the array is already sorted

    for (int i = 0; i < n - 1; i++)
    {
        int minIndex = i;

        // minimum element in the unsorted portion
        for (int j = i + 1; j < n; j++)
        {
            if (arr[j] < arr[minIndex])
```

**Data Structures Applications Laboratory (21EECF201/23EVTF203)**

```
{
    minIndex = j;
    isSorted = 0; // If smaller element, array isn't sorted
}
}

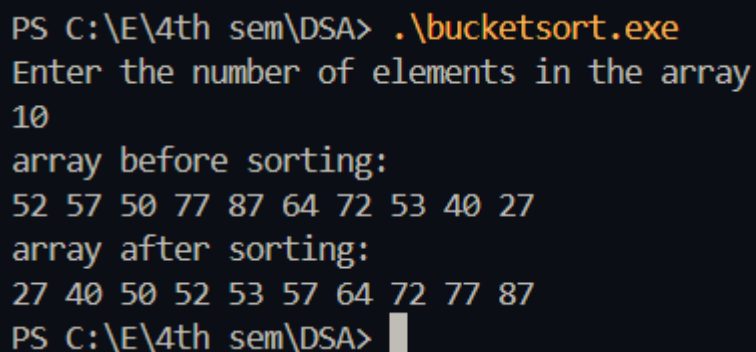
// Swap only if needed
if (minIndex != i) {
    int temp = arr[minIndex];
    arr[minIndex] = arr[i];
    arr[i] = temp;
}

// If no swaps were needed, the array is already sorted
if (isSorted) {
    printf("Array is already sorted, exiting early.\n"); // Debug message
    return;
}

isSorted = 1; // Reset sorted flag
}
}
```

Shellsort:

Output: Put screenshots of the output



```
PS C:\E\4th sem\DSA> .\bucketsort.exe
Enter the number of elements in the array
10
array before sorting:
52 57 50 77 87 64 72 53 40 27
array after sorting:
27 40 50 52 53 57 64 72 77 87
PS C:\E\4th sem\DSA>
```

**Data Structures Applications Laboratory (21EECF201/23EVTf203)**

```
PS C:\E\4th sem\DSA> .\countingsort.exe
```

```
array before sorting:
```

```
52 57 50 77 87 64 72 53 40 27
```

```
array after sorting:
```

```
27 40 50 52 53 57 64 72 77 87
```

```
PS C:\E\4th sem\DSA> .\bubblesort.exe
```

```
Loading numbers from file...
```

```
Before sorting:
```

```
33 53 24 37 44 83 41 68 10 40
```

```
Sorting in progress...
```

```
Sorted array:
```

```
10 24 33 37 40 41 44 53 68 83
```

```
PS C:\E\4th sem\DSA>
```

```
PS C:\E\4th sem\DSA> .\heapsort.exe
```

```
array before sorting:
```

```
52 57 50 77 87 64 72 53 40 27
```

```
array after sorting:
```

```
27 40 50 52 53 57 64 72 77 87
```

```
PS C:\E\4th sem\DSA>
```

```
PS C:\E\4th sem\DSA> .\insertionSort.exe
```

```
array before sorting:
```

```
52 57 50 77 87 64 72 53 40 27
```

```
array after sorting:
```

```
27 40 50 52 53 57 64 72 77 87
```

```
PS C:\E\4th sem\DSA>
```

```
PS C:\E\4th sem\DSA> .\mergesort.exe
```

```
array before sorting:
```

```
52 57 50 77 87 64 72 53 40 27
```

```
array after sorting:
```

```
27 40 50 52 53 57 64 72 77 87
```

```
PS C:\E\4th sem\DSA>
```

```
PS C:\E\4th sem\DSA> .\quicksort.exe
```

```
array before sorting:
```

```
52 57 50 77 87 64 72 53 40 27
```

```
array after sorting:
```

```
27 40 50 52 53 57 64 72 77 87
```

## Data Structures Applications Laboratory (21EECF201/23EVTf203)

```
PS C:\E\4th sem\DSA> gcc .\radixSort.c -o radixSort
PS C:\E\4th sem\DSA> .\radixSort.exe
array before sorting:
52 57 50 77 87 64 72 53 40 27
array after sorting:
27 40 50 52 53 57 64 72 77 87
PS C:\E\4th sem\DSA>
```

```
PS C:\E\4th sem\DSA> .\selectionSort.exe
array before sorting:
52 57 50 77 87 64 72 53 40 27
array after sorting:
27 40 50 52 53 57 64 72 77 87
```

```
PS C:\E\4th sem\DSA> .\shellSort.exe
array before sorting:
52 57 50 77 87 64 72 53 40 27
array after sorting:
27 40 50 52 53 57 64 72 77 87
```

### Task 4:

Sorting algorithm	Time complexity		
	Best case	Average case	Worst case
Quick Sort	$O(N \log N)$	$O(N \log N)$	$O(N^2)$
Bubble Sort	$O(N)$	$O(N^2)$	$O(N^2)$
Merge Sort	$O(N \log N)$	$O(N \log N)$	$O(N \log N)$
Selection sort	$N^2$	$N^2$	$N^2$
Insertion Sort	$O(N)$	$O(N^2)$	$O(N^2)$
Heap Sort	$O(N \log N)$	$O(N \log N)$	$O(N \log N)$
Counting Sort	$O(N+K)$	$O(N+K)$	$O(N+K)$
Shell Sort	$O(N \log N)$	$O(N \log N)$	$O(N^2)$
Tim Sort	$O(N \log N)$	$O(N \log N)$	$O(N \log N)$
Radix Sort	$O(d*(N+b))$	$O(d*(N+b))$	$O(d*(N+b))$