**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 |

|  |
| --- |
| **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(N2), 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 N2, 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 N2.  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 sorted array where it belongs to by iterating constantly and check if the number should be between them. Worst case: O(N2), 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(N2), 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:  Quick sort:  #include<stdio.h>  #include<stdlib.h>  *void* readFile(*int* arr*[]*,*int* n, *char* \*filename);  *void* quicksort(*int* arr*[]*,*int* low,*int* high);  *int* main()  {  *int* n=10,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");      quicksort(arr,0,n-1);      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);      }  }  *void* quicksort(*int* arr*[]*,*int* low, *int* high)  {      if(low<high)      {  *int* a=partition(arr,low,high);          quicksort(arr,low,a-1);          quicksort(arr,a+1,high);      }  }  *int* partition(*int* arr*[]*,*int* low,*int* high)  {  *int* pivot=arr[high];  *int* i=low-1;      for(*int* j=low;j<high;j++)      {          if(arr[j]<pivot)          {              i++;  *int* temp=arr[i];              arr[i]=arr[j];              arr[j]=temp;          }      }  *int* temp=arr[i+1];      arr[i+1]=arr[high];      arr[high]=temp;      return i+1;  }  Bubblesort: #include <stdio.h>  #include <stdlib.h>  *void* readFile(*int* arr*[]*,*int* n, *char* \*filename);  *void* bubbleSort(*int* arr*[]*, *int* n);  *int* main()  {  *int* n=10;  *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");        bubbleSort(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);      }  }  *void* bubbleSort(*int* arr*[]*, *int* n)  {  *int* sorted;      for(*int* i=0;i<n-1;i++)      {          sorted = 1;          for(*int* j=0;j<n-i-1;j++)          {              if(arr[j]>arr[j+1])              {  *int* temp=arr[j];                  arr[j]=arr[j+1];                  arr[j+1]=temp;                  sorted=0;              }          }          if(sorted)              break;      }  }  Insertion sort: |
| Output: Put screenshots of the output |

**Task 4:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Time complexity** | | |
| **Sorting algorithm** | **Best case** | **Average case** | **Worst case** |
| **Quick Sort** | O(N log N) |  | O(N2) |
| **Bubble Sort** | O(N2) |  | O(N) |
| **Merge Sort** | O(N log N) | O(N log N) | O(N log N) |
| **Selection sort** | N2 | N2 | N2 |
| **Insertion Sort** | O(N2) |  | O(N) |
| **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(N2) |
| **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)) |