**Name of the student: Varun Vivek Pai**

**USN: 01FE23BEC126**

**Div: C**

**Roll No: 332**

|  |
| --- |
| **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>  void random(int N, int P, int Q)  {      int num, i;      FILE \*file = fopen("Input.txt", "w");      if (file)      {            srand(time(NULL));          for (i = 0; i < N; i++)          {              num = P + rand() % (Q - P + 1);              fprintf(file, "%d\n", num);          }          fclose(file);          printf("Successfully written %d random numbers to Input.txt\n", N);      }      else          printf("Error opening file!\n");  }  int main()  {      int N, P, Q;      printf("Enter the number of random numbers to generate (N): ");      scanf("%d", &N);      printf("Enter the lower bound (P): ");      scanf("%d", &P);      printf("Enter the upper bound (Q): ");      scanf("%d", &Q);      if (P > Q)          printf("Invalid range! P should be less than or equal to Q.\n");      else          random(N, P, Q);      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.Bubble Sort  2. Selection Sort  3. Insertion Sort  4.Merge Sort  5. Quick Sort  6. Heap Sort  7. Shell Sort  8.Counting Sort  9.Radix Sort  10. Bucket Sort   1. **Bubble Sort**: Repeatedly compares and swaps adjacent elements if they are in the wrong order, effectively "bubbling" larger elements to the end of the list. 2. **Selection Sort**: Identifies the smallest (or largest) element from the unsorted portion and swaps it with the first unsorted element, moving the boundary of the sorted section one element forward. 3. **Insertion Sort**: Builds a sorted list by taking each element from the unsorted portion and inserting it into its correct position within the sorted portion. 4. **Merge Sort**: Recursively divides the list into halves until each sublist contains a single element, then merges these sublists in a sorted manner to produce the final sorted list. 5. **Quick Sort**: Selects a pivot element, partitions the list into elements less than and greater than the pivot, and recursively applies the same process to the sublists. 6. **Heap Sort**: Transforms the list into a heap structure, repeatedly extracts the maximum (or minimum) element, and reconstructs the heap until the list is sorted. 7. **Shell Sort**: An extension of insertion sort that allows the exchange of items that are far apart by comparing elements separated by a gap, which decreases over successive passes. 8. **Counting Sort**: Counts the occurrences of each distinct element, calculates their positions, and places them into the output array accordingly, making it efficient for sorting integers within a specific range. 9. **Radix Sort**: Processes each digit of the numbers from least significant to most significant, grouping them by each digit, and combining the groups to form a sorted list. 10. **Bucket Sort**: Distributes elements into several 'buckets', sorts each bucket individually (often using another sorting algorithm), and then concatenates the sorted buckets to form the final sorted list. |

|  |
| --- |
| **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:   1. **Bubble Sort:**   #include <stdio.h>  #include <stdlib.h>  #define MAX\_SIZE 1000  void bubbleSort(int arr[], int n)  {      int temp, i, j;      for (i = 0; i < n - 1; i++) {          for (j = 0; j < n - i - 1; j++) {              if (arr[j] > arr[j + 1]) {                  temp = arr[j];                  arr[j] = arr[j + 1];                  arr[j + 1] = temp;              }          }      }  }  int readNum(const char \*filename, int arr[])  {      FILE \*file = fopen(filename, "r");      int count = 0;      if (file)      {          while (fscanf(file, "%d", &arr[count]) != EOF && count < MAX\_SIZE)              count++;          fclose(file);      }      else          printf("Error opening file: %s\n", filename);      return count;  }  void writeFile(const char \*filename, int arr[], int n)  {      FILE \*file = fopen(filename, "w");      int count = 0, i;      if (file)      {          for (i = 0; i < n; i++)              fprintf(file, "%d\n", arr[i]);          fclose(file);          printf("Sorted numbers written to %s\n", filename);      }      else          printf("Error opening file: %s\n", filename);  }  int main()  {      int numbers[MAX\_SIZE], count;      count = readNum("Input.txt", numbers);      if (count)      {          bubbleSort(numbers, count);          writeFile("Sorted.txt", numbers, count);          printf("Sorting complete. Check Sorted.txt for results.\n");      }      else          printf("File is empty");      return 0;  }   1. **Selection Sort:**   #include <stdio.h>  #include <stdlib.h>  #define MAX\_SIZE 1000  void selectionSort(int arr[], int n)  {      int i, j, min, temp;      for (i = 0; i < n - 1; i++)      {          min = i;          for (j = i + 1; j < n; j++)          {              if (arr[j] < arr[min])                  min = j;          }          temp = arr[i];          arr[i] = arr[min];          arr[min] = temp;      }  }  int readNum(const char \*filename, int arr[])  {      FILE \*file = fopen(filename, "r");      int count = 0;      if (file)      {          while (fscanf(file, "%d", &arr[count]) != EOF && count < MAX\_SIZE)              count++;          fclose(file);      }      else          printf("Error opening file: %s\n", filename);      return count;  }  void writeFile(const char \*filename, int arr[], int n)  {      FILE \*file = fopen(filename, "w");      int i;      if (file)      {          for (i = 0; i < n; i++)              fprintf(file, "%d\n", arr[i]);          fclose(file);          printf("Sorted numbers written to %s\n", filename);      }      else          printf("Error opening file: %s\n", filename);  }  int main()  {      int numbers[MAX\_SIZE], count;      count = readNum("Input.txt", numbers);      if (count)      {          selectionSort(numbers, count);          writeFile("Sorted.txt", numbers, count);          printf("Sorting complete. Check Sorted.txt for results.\n");      }      else          printf("File is empty");      return 0;  }   1. **Insertion Sort:**   #include <stdio.h>  #include <stdlib.h>  #define MAX\_SIZE 1000  void insertionSort(int arr[], int n)  {      int i, j, key;      for (i = 1; i < n; i++)      {          key = arr[i];          for (j = i - 1; j >= 0 && arr[j] > key; j--)              arr[j + 1] = arr[j];          arr[j + 1] = key;      }  }  int readNum(const char \*filename, int arr[])  {      FILE \*file = fopen(filename, "r");      int count = 0;      if (file)      {          while (fscanf(file, "%d", &arr[count]) != EOF && count < MAX\_SIZE)              count++;          fclose(file);      }      else          printf("Error opening file: %s\n", filename);      return count;  }  void writeFile(const char \*filename, int arr[], int n)  {      FILE \*file = fopen(filename, "w");      int i;      if (file)      {          for (i = 0; i < n; i++)              fprintf(file, "%d\n", arr[i]);          fclose(file);          printf("Sorted numbers written to %s\n", filename);      }      else          printf("Error opening file: %s\n", filename);  }  int main()  {      int numbers[MAX\_SIZE], count;      count = readNum("Input.txt", numbers);      if (count)      {          insertionSort(numbers, count);          writeFile("Sorted.txt", numbers, count);          printf("Sorting complete. Check Sorted.txt for results.\n");      }      else          printf("File is empty");      return 0;  }   1. **Merge Sort:**   #include <stdio.h>  #include <stdlib.h>  #define MAX\_SIZE 1000  void merge(int arr[], int left, int mid, int right)  {      int i, j, k, n1 = mid - left + 1, n2 = right - mid, L[n1], R[n2];      for (i = 0; i < n1; i++)          L[i] = arr[left + i];      for (j = 0; j < n2; j++)          R[j] = arr[mid + 1 + j];      for (i = j = 0, k = left; i < n1 && j < n2;)      {          if (L[i] <= R[j])              arr[k++] = L[i++];          else              arr[k++] = R[j++];      }      while (i < n1)          arr[k++] = L[i++];      while (j < n2)          arr[k++] = R[j++];  }  void mergeSort(int arr[], int left, int right)  {      int mid;      if (left < right)      {          mid = left + (right - left) / 2;          mergeSort(arr, left, mid);          mergeSort(arr, mid + 1, right);          merge(arr, left, mid, right);      }  }  int readNum(const char \*filename, int arr[])  {      FILE \*file = fopen(filename, "r");      int count = 0;      if (file)      {          while (fscanf(file, "%d", &arr[count]) != EOF && count < MAX\_SIZE)              count++;          fclose(file);      }      else          printf("Error opening file: %s\n", filename);      return count;  }  void writeFile(const char \*filename, int arr[], int n)  {      FILE \*file = fopen(filename, "w");      int i;      if (file)      {          for (i = 0; i < n; i++)              fprintf(file, "%d\n", arr[i]);          fclose(file);          printf("Sorted numbers written to %s\n", filename);      }      else          printf("Error opening file: %s\n", filename);  }  int main()  {      int numbers[MAX\_SIZE], count;      count = readNum("Input.txt", numbers);      if (count)      {          mergeSort(numbers, 0, count - 1);          writeFile("Sorted.txt", numbers, count);          printf("Sorting complete. Check Sorted.txt for results.\n");      }      else          printf("File is empty");      return 0;  }   1. **Quick sort:**   #include <stdio.h>  #include <stdlib.h>  #define MAX\_SIZE 1000  int partition(int arr[], int low, int high)  {      int pivot = arr[high], i = low - 1, j, temp;      for (j = low; j < high; j++)      {          if (arr[j] < pivot)          {              i++;              temp = arr[i];              arr[i] = arr[j];              arr[j] = temp;          }      }      temp = arr[i + 1];      arr[i + 1] = arr[high];      arr[high] = temp;      return (i + 1);  }  void quickSort(int arr[], int low, int high)  {      int pi;      if (low < high)      {          pi = partition(arr, low, high);          quickSort(arr, low, pi - 1);          quickSort(arr, pi + 1, high);      }  }  int readNum(const char \*filename, int arr[])  {      FILE \*file = fopen(filename, "r");      int count = 0;      if (file)      {          while (fscanf(file, "%d", &arr[count]) != EOF && count < MAX\_SIZE)              count++;          fclose(file);      }      else          printf("Error opening file: %s\n", filename);      return count;  }  void writeFile(const char \*filename, int arr[], int n)  {      FILE \*file = fopen(filename, "w");      int i;      if (file)      {          for (i = 0; i < n; i++)              fprintf(file, "%d\n", arr[i]);          fclose(file);          printf("Sorted numbers written to %s\n", filename);      }      else          printf("Error opening file: %s\n", filename);  }  int main()  {      int numbers[MAX\_SIZE], count;      count = readNum("Input.txt", numbers);      if (count)      {          quickSort(numbers, 0, count - 1);          writeFile("Sorted.txt", numbers, count);          printf("Sorting complete. Check Sorted.txt for results.\n");      }      else          printf("File is empty");      return 0;  }   1. **Heap Sort:**   #include <stdio.h>  #include <stdlib.h>  #define MAX\_SIZE 1000  void heapify(int arr[], int n, int i)  {      int largest = i, left = 2 \* i + 1, right = 2 \* i + 2;      if (left < n && arr[left] > arr[largest])          largest = left;      if (right < n && arr[right] > arr[largest])          largest = right;      if (largest != i)      {          int temp = arr[i];          arr[i] = arr[largest];          arr[largest] = temp;          heapify(arr, n, largest);      }  }  void heapSort(int arr[], int n)  {      int i, temp;      for (i = n / 2 - 1; i >= 0; i--)          heapify(arr, n, i);      for (i = n - 1; i > 0; i--)      {          temp = arr[0];          arr[0] = arr[i];          arr[i] = temp;          heapify(arr, i, 0);      }  }  int readNum(const char \*filename, int arr[])  {      FILE \*file = fopen(filename, "r");      int count = 0;      if (file)      {          while (fscanf(file, "%d", &arr[count]) != EOF && count < MAX\_SIZE)              count++;          fclose(file);      }      else          printf("Error opening file: %s\n", filename);      return count;  }  void writeFile(const char \*filename, int arr[], int n)  {      FILE \*file = fopen(filename, "w");      int i;      if (file)      {          for (i = 0; i < n; i++)              fprintf(file, "%d\n", arr[i]);          fclose(file);          printf("Sorted numbers written to %s\n", filename);      }      else          printf("Error opening file: %s\n", filename);  }  int main()  {      int numbers[MAX\_SIZE], count;      count = readNum("Input.txt", numbers);      if (count)      {          heapSort(numbers, count);          writeFile("Sorted.txt", numbers, count);          printf("Sorting complete. Check Sorted.txt for results.\n");      }      else          printf("File is empty");      return 0;  }   1. **Shell Sort:**   #include <stdio.h>  #include <stdlib.h>  #define MAX\_SIZE 1000  void shellSort(int arr[], int n)  {      int i, j, temp;      for (int gap = n / 2; gap > 0; gap /= 2)      {          for (i = gap; i < n; i++)          {              temp = arr[i];              for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)                  arr[j] = arr[j - gap];              arr[j] = temp;          }      }  }  int readNum(const char \*filename, int arr[])  {      FILE \*file = fopen(filename, "r");      int count = 0;      if (file)      {          while (fscanf(file, "%d", &arr[count]) != EOF && count < MAX\_SIZE)              count++;          fclose(file);      }      else          printf("Error opening file: %s\n", filename);      return count;  }  void writeFile(const char \*filename, int arr[], int n)  {      FILE \*file = fopen(filename, "w");      int i;      if (file)      {          for (i = 0; i < n; i++)              fprintf(file, "%d\n", arr[i]);          fclose(file);          printf("Sorted numbers written to %s\n", filename);      }      else          printf("Error opening file: %s\n", filename);  }  int main()  {      int numbers[MAX\_SIZE], count;      count = readNum("Input.txt", numbers);      if (count)      {          shellSort(numbers, count);          writeFile("Sorted.txt", numbers, count);          printf("Sorting complete. Check Sorted.txt for results.\n");      }      else          printf("File is empty");      return 0;  }   1. **Counting Sort:**   #include <stdio.h>  #include <stdlib.h>  #define MAX\_SIZE 1000  int findMax(int arr[], int n)  {      int max = arr[0], i;      for (i = 1; i < n; i++)      {          if (arr[i] > max)              max = arr[i];      }      return max;  }  void countingSort(int arr[], int n)  {      int max, i, \*count, \*output;      max =  findMax(arr, n);      count = calloc(max + 1, sizeof(int));      output = malloc(n \* sizeof(int));      for (i = 0; i < n; i++)          count[arr[i]]++;      for (i = 1; i <= max; i++)          count[i] += count[i - 1];      for (i = n - 1; i >= 0; i--)      {          output[count[arr[i]] - 1] = arr[i];          count[arr[i]]--;      }      for (i = 0; i < n; i++)          arr[i] = output[i];      free(count);      free(output);  }  int readNum(const char \*filename, int arr[])  {      FILE \*file = fopen(filename, "r");      int count = 0;      if (file)      {          while (fscanf(file, "%d", &arr[count]) != EOF && count < MAX\_SIZE)              count++;          fclose(file);      }      else          printf("Error opening file: %s\n", filename);      return count;  }  void writeFile(const char \*filename, int arr[], int n)  {      FILE \*file = fopen(filename, "w");      int i;      if (file)      {          for (i = 0; i < n; i++)              fprintf(file, "%d\n", arr[i]);          fclose(file);          printf("Sorted numbers written to %s\n", filename);      }      else          printf("Error opening file: %s\n", filename);  }  int main()  {      int numbers[MAX\_SIZE], count;      count = readNum("Input.txt", numbers);      if (count)      {          countingSort(numbers, count);          writeFile("Sorted.txt", numbers, count);          printf("Sorting complete. Check Sorted.txt for results.\n");      }      else          printf("File is empty");      return 0;  }   1. **Radix Sort:**   #include <stdio.h>  #include <stdlib.h>  #define MAX\_SIZE 1000  int findMax(int arr[], int n)  {      int max = arr[0], i;      for (i = 1; i < n; i++)      {          if (arr[i] > max)              max = arr[i];      }      return max;  }  void countingSortByDigit(int arr[], int n, int exp)  {      int output[n], count[10] = {0}, i;      for (i = 0; i < n; i++)          count[(arr[i] / exp) % 10]++;      for (i = 1; i < 10; i++)          count[i] += count[i - 1];      for (i = n - 1; i >= 0; i--)      {          output[count[(arr[i] / exp) % 10] - 1] = arr[i];          count[(arr[i] / exp) % 10]--;      }      for (i = 0; i < n; i++)          arr[i] = output[i];  }  void radixSort(int arr[], int n)  {      int max = findMax(arr, n), exp;      for (int exp = 1; max / exp > 0; exp \*= 10)          countingSortByDigit(arr, n, exp);  }  int readNum(const char \*filename, int arr[])  {      FILE \*file = fopen(filename, "r");      int count = 0;      if (file)      {          while (fscanf(file, "%d", &arr[count]) != EOF && count < MAX\_SIZE)              count++;          fclose(file);      }      else          printf("Error opening file: %s\n", filename);      return count;  }  void writeFile(const char \*filename, int arr[], int n)  {      FILE \*file = fopen(filename, "w");      int i;      if (file)      {          for (i = 0; i < n; i++)              fprintf(file, "%d\n", arr[i]);          fclose(file);          printf("Sorted numbers written to %s\n", filename);      }      else          printf("Error opening file: %s\n", filename);  }  int main()  {      int numbers[MAX\_SIZE], count;      count = readNum("Input.txt", numbers);      if (count)      {          radixSort(numbers, count);          writeFile("Sorted.txt", numbers, count);          printf("Sorting complete. Check Sorted.txt for results.\n");      }      else          printf("File is empty");      return 0;  }   1. **Bucket Sort:**   #include <stdio.h>  #include <stdlib.h>  #define MAX\_SIZE 1000  #define BUCKET\_SIZE 10  typedef struct {      int \*array, count;  } Bucket;  int findMax(int arr[], int n)  {      int max = arr[0], i;      for (i = 1; i < n; i++)      {          if (arr[i] > max)              max = arr[i];      }      return max;  }  void insertionSort(int arr[], int n)  {      int i, j, key;      for (i = 1; i < n; i++)      {          for (key = arr[i], j = i - 1; j >= 0 && arr[j] > key; j--)              arr[j + 1] = arr[j];          arr[j + 1] = key;      }  }  void bucketSort(int arr[], int n)  {      int max, min, i, j, range, bucketIndex, index;      Bucket buckets[BUCKET\_SIZE];      if (n >= 0)      {          max = findMax(arr, n);          min = arr[0];          for (i = 1; i < n; i++)          {              if (arr[i] < min)                  min = arr[i];          }          range = (max - min) / BUCKET\_SIZE + 1;          for (i = 0; i < BUCKET\_SIZE; i++)          {              buckets[i].array = (int \*)malloc(n \* sizeof(int));              buckets[i].count = 0;          }          for (i = 0; i < n; i++)          {              bucketIndex = (arr[i] - min) / range;              buckets[bucketIndex].array[buckets[bucketIndex].count++] = arr[i];          }          for (index = i = 0; i < BUCKET\_SIZE; i++)          {              insertionSort(buckets[i].array, buckets[i].count);              for (j = 0; j < buckets[i].count; j++)                  arr[index++] = buckets[i].array[j];              free(buckets[i].array);          }      }      else          printf("No numbers to sort\n");  }  int readNum(const char \*filename, int arr[])  {      FILE \*file = fopen(filename, "r");      int count = 0;      if (file)      {          while (fscanf(file, "%d", &arr[count]) != EOF && count < MAX\_SIZE)              count++;          fclose(file);      }      else          printf("Error opening file: %s\n", filename);      return count;  }  void writeFile(const char \*filename, int arr[], int n)  {      FILE \*file = fopen(filename, "w");      int i;      if (file)      {          for (i = 0; i < n; i++)              fprintf(file, "%d\n", arr[i]);          fclose(file);          printf("Sorted numbers written to %s\n", filename);      }      else          printf("Error opening file: %s\n", filename);  }  int main()  {      int numbers[MAX\_SIZE], count;      count = readNum("Input.txt", numbers);      if (count)      {          bucketSort(numbers, count);          writeFile("Sorted.txt", numbers, count);          printf("Sorting complete. Check Sorted.txt for results.\n");      }      else          printf("File is empty");      return 0;  } |
| Output: Put screenshots of the output  Bubble Sort:        Selection Sort:        Insertion Sort:        Merge Sort:        Quick Sort:        Heap Sort:        Shell sort:        Counting Sort:        Radix Sort:        Bucket Sort: |

**Task 4:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Time complexity** | | |
| **Sorting algorithm** | **Best case** | **Average case** | **Worst case** |
| Bubble Sort | **O(n2)** | **O(n2)** | **O(n2)** |
| Selection Sort | **O(n2)** | **O(n2)** | **O(n2)** |
| Insertion Sort | **O(n)** | **O(n2)** | **O(n2)** |
| Merge Sort | **O(n log n)** | **O(n log n)** | **O(n2)** |
| Quick Sort | **O(n log n)** | **O(n log n)** | **O(n log n)** |
| Heap Sort | **O(n log n)** | **O(n log n)** | **O(n log n)** |
| Shell Sort | **O(n log n)** | **O(n log2 n)** | **O(n2)** |
| Counting Sort | **O(n + k)** | **O(n + k)** | **O(n + k)** |
| Radix Sort | **O(nk)** | **O(nk)** | **O(nk)** |
| Bucket Sort | **O(n + k)** | **O(n + k)** | **O(n2)** |