|  |  |
| --- | --- |
| **NAME:** | Shubham Solanki |
| **UID:** | 2022301015 |
| **SUBJECT** | Design and Analysis of Algorithms |
| **EXPERIMENT NO:** | 2(B) |
| **AIM:** | Understanding more concepts regarding quick sort algorithm |
| **Algorithm:** | **Quick Sort Algorithm**  partition (arr[], low, high)  { // pivot (Element to be placed at right position)  pivot = arr[high];  i = (low – 1) // Index of smaller element and indicates the  // right position of pivot found so far  for (j = low; j <= high- 1; j++){  // If current element is smaller than the pivot  if (arr[j] < pivot){  i++; // increment index of smaller element  swap arr[i] and arr[j]  }  }  swap arr[i + 1] and arr[high])  return (i + 1)  }  quickSort(arr[], low, high) {  if (low < high) {  /\* pi is partitioning index, arr[pi] is now at right place \*/  pi = partition(arr, low, high);  quickSort(arr, low, pi – 1); // Before pi  quickSort(arr, pi + 1, high); // After pi  } |
| **Code:** | #include <stdio.h>  #include <stdlib.h>  #include <time.h>  long SWAP = 0;  void merge(int arr[], int p, int q, int r) {    int i, j, k;    int n1 = q - p + 1;    int n2 = r - q;    int L[n1], R[n2];    for (i = 0; i < n1; i++)      L[i] = arr[p + i];    for (j = 0; j < n2; j++)      R[j] = arr[q + 1 + j];    i = 0;    j = 0;    k = p;    while (i < n1 && j < n2) {      if (L[i] <= R[j]) {        arr[k] = L[i];        i++;      } else {        arr[k] = R[j];        j++;      }      k++;    }    while (i < n1) {      arr[k] = L[i];      i++;      k++;    }    while (j < n2) {      arr[k] = R[j];      j++;      k++;    }  }  void mergeSort(int arr[], int l, int r)  {    if (l < r) {      int m = l + (r - l) / 2;      mergeSort(arr, l, m);      mergeSort(arr, m + 1, r);      merge(arr, l, m, r);    }  }  int quicksort(int a[], int start, int end) {    int pivot = a[end];  *//int pivot = a[start];*  *//int random = start + rand() % (end - start);*  *//int pivot = a[random];*  *//int mid = start + (end - start)/2;*  *//int pivot = a[mid];*    int i = (start - 1);    for (int j = start; j <= end - 1; j++) {      if (a[j] < pivot) {        i++;        int t = a[i];        a[i] = a[j];        a[j] = t;        SWAP++;      }    }    int t = a[i + 1];    a[i + 1] = a[end];    a[end] = t;    SWAP++;    return (i + 1);  }  double quick(int a[], int start, int end) {    if (start < end) {      int p = quicksort(a, start, end);      quick(a, start, p - 1);      quick(a, p + 1, end);    }  }  int main() {    double qust, mest;    srand(time(0));    FILE \* fp, \* file;    fp = fopen("random.txt", "w");    for (int i = 0; i < 100000; i++) {      fprintf(fp, "%d\n", rand() % 900001 + 100000);    }    int upper\_limit = 100;    fclose(fp);    file = fopen("outputEnd.txt", "w");    fprintf(file, "Block\tMerSort\tQuickSort\tSwaps\n");    for (int i = 0; i < 1000; i++) {      fp = fopen("random.txt", "r");      int arr1[upper\_limit], arr2[upper\_limit], temp\_num;      for (int j = 0; j < upper\_limit; j++) {        fscanf(fp, "%d", & temp\_num);        arr1[j] = temp\_num;        arr2[j] = temp\_num;      }      fclose(fp);      clock\_t t;      t = clock();      mergeSort(arr2, 0, upper\_limit - 1);      t = clock() - t;      mest = ((double) t) / CLOCKS\_PER\_SEC;      clock\_t t1;      t1 = clock();      qust = quick(arr1, 0, upper\_limit - 1);      t1 = clock() - t1;      qust = ((double) t1) / CLOCKS\_PER\_SEC;      fprintf(file, "%d\t%lf\t%lf\t%ld\n", i + 1, mest, qust, SWAP);      fflush(stdout);      upper\_limit += 100;    }    return 0;  } |
| **Graphs and Observation:** | **Running time comparison for Different Pivot Positions**     * Here, we can see that the time complexity of quick sort is nearly the same even when varied pivot points are taken into account. * We can see that a quick sort when the pivot is at a random position takes longer than when the pivot is in an end position at the conclusion of execution. * Despite the fact that both executions are finished in 0.1 seconds.   **Number of swaps considering different pivot positions**     * We can see from this that fewer swaps are needed for a quick sort when the pivot is in a random position as opposed to when it is at the end. * The average number of swaps is over 500,000,000. * Throughout the entire execution, the number of swaps keeps rising. |
| **Conclusion:** | Thus, we have provided observations for different pivots for quick sort algorithms. |