|  |  |
| --- | --- |
| NAME: | Darsh Shah |
| UID No. | 2021700057 |
| BRANCH: | SY CSE DS |
| BATCH: | D4 |
| SUBJECT | DAA |
| EXPERIMENT No. | 2 |
| DATE: | 19th February , 2023 |

|  |  |
| --- | --- |
| AIM: | Experiment based on divide and conquer approach. |
| Program 1 | |
| PROBLEM STATEMENT : | Quicksort– It picks an element called as pivot, and then it partitions the given array around the picked pivot element. It then arranges the entire array in two sub-array such that one array holds values that are smaller than the specified value (Pivot), and another array holds the values that are greater than the pivot.The Divide and Conquer steps of Quicksort perform following functions. Divide: In Divide, first pick a pivot element. After that, partition or rearrange the array into two sub-arrays such that each element in the left sub-array is less than or equal to the pivot element and each element in the right sub-array is larger than the pivot element. Conquer: Recursively, sort two subarrays with Quicksort Combine: Combine the already sorted array.  Merge sort– Merge sort is similar to the quick sort algorithm as it uses the divide and conquer approach to sort the elements. It divides the given list into two equal halves, calls itself for the two halves and then merges the two sorted halves. We have to define the merge() function to perform the merging.The sub-lists are divided again and again into halves until the list cannot be divided further. Then we combine the pair of one element lists into twoelement lists, sorting them in the process. The sorted two-element pairs is merged into the four-element lists, and so on until we get the sorted list. |
| GRAPHS: | Merge sort graph  Y  X  Quicksort graph  Y  X  Observations:  Both graphs are plotted in excel.  Merge sort is more efficient than quicksort.  Time consumed for Merge sort is lesser compared to quicksort.  Maximum time in merge sort goes to around 0.025 sec  Maximum time for quick sort is near 2.25 sec.  The slope of merge sort is higher, more changed values compared to quicksort |
| PROGRAM: | #include <stdio.h>  #include <time.h>  #include <stdlib.h>  double populate(int a[], int b[], int n) {  clock\_t start, end;  double cpu\_time\_used;  start = clock();  for(int i = 0; i < n; i++)  {  int r = rand();  a[i] = b[i] = r;  }  end = clock();  FILE \*fp = fopen("./random.txt", "w+");  if(!fp) {  printf("Error opening file\n");  return -1;  }  for(int i = 0; i < n; i++) {  fprintf(fp, "%d\n", a[i]);  }  cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;  return cpu\_time\_used;  }  void merge(int a[], int l, int m, int r) {  int i, j, k;  int n1 = m - l + 1;  int n2 = r - m;  int L[n1], R[n2];  /\* Copy data to temp arrays L[] and R[] \*/  for (i = 0; i < n1; i++)  L[i] = a[l + i];  for (j = 0; j < n2; j++)  R[j] = a[m + 1 + j];  /\* Merge the temp arrays back into arr[l..r]\*/  i = j = 0;  k = l;  while(i < n1 && j < n2) {  if(L[i] <= R[j]) {  a[k] = L[i];  i++;  }  else {  a[k] = R[j];  j++;  }  k++;  }  /\* Copy the remaining elements of L[], if there  are any \*/  while (i < n1) {  a[k] = L[i];  i++;  k++;  }  /\* Copy the remaining elements of R[], if there  are any \*/  while (j < n2) {  a[k] = R[j];  j++;  k++;  }  }  void mergeSort(int a[], int l, int r) {  // printf("\nl: %d, r: %d\n", l, r);  // printf("\nO l: %d, r: %d\n", l, r);  if(l<r) {  int m = (l+r)/2;  //call for left array  // printf("\nI l: %d, m: %d, r: %d\n", l, m, r);  mergeSort(a, l, m);    //call for right array  mergeSort(a, m+1, r);  //merge the two arrays  merge(a, l, m, r);  }  }  double mergeCalc(int a[], int n) {  FILE \*fp = fopen("./mergeSort.csv", "w+");  // printf("File opened\n");  double totalTime = 0;  if(!fp) {  printf("Error opening file\n");  return -1;  }  fprintf(fp, "n, time\n");  for (int i = 99; i <= n; i+=100)  {  clock\_t start, end;  double cpu\_time\_used;  start = clock();  mergeSort(a, 0, i);  end = clock();  cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;  totalTime += cpu\_time\_used;  fprintf(fp, "%d, %f\n", i+1, cpu\_time\_used);  printf("Sorted from 0 to %d in %.2fs\n", i, cpu\_time\_used);  }  fclose(fp);  fp = fopen("./mergeSort.txt", "w+");  for(int i = 0; i < n; i++) {  fprintf(fp, "%d\n", a[i]);  }  fclose(fp);  return totalTime;  }  void swap(int \*x, int \*y) {  int t = \*x;  \*x = \*y;  \*y = t;  }  int partition(int arr[], int low, int high)  {  int pivot = arr[high]; // pivot  int i  = (low  - 1); // Index of smaller element and indicates  // the right position of pivot found so far    for (int 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], &arr[j]);  }  }  swap(&arr[i + 1], &arr[high]);  return (i + 1);  }    void quickSort(int a[], int low, int high)  {  if (low < high) {  /\* pi is partitioning index, arr[p] is now  at right place \*/  int pi = partition(a, low, high);    // Separately sort elements before  // partition and after partition  quickSort(a, low, pi - 1);  quickSort(a, pi + 1, high);  }  }  double qC(int a[], int n) {  FILE \*fp = fopen("./quickSort.csv", "w+");  // printf("File opened\n");  double totalTime = 0;  if(!fp) {  printf("Error opening file\n");  return -1;  }  fprintf(fp, "n, time\n");  for (int i = 99; i <= n; i+=100)  {  clock\_t start, end;  double cpu\_time\_used;  start = clock();  quickSort(a, 0, i);  end = clock();  cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;  totalTime += cpu\_time\_used;  fprintf(fp, "%d, %f\n", i+1, cpu\_time\_used);  printf("Sorted from 0 to %d in %.2fs\n", i, cpu\_time\_used);  }  fclose(fp);  fp = fopen("./quickSort.txt", "w+");  for(int i = 0; i < n; i++) {  fprintf(fp, "%d\n", a[i]);  }  fclose(fp);  return totalTime;  }  void printArr(int a[], int n) {  for (int i = 0; i <=n; i++)  printf("%d\n", a[i]);  }  int main()  {  int n = 100000;  int a[n],b[n];  double timeToPopulate = populate(a, b, n);  printf("Time taken to populate: %f\nSorting...\n", timeToPopulate);  //msort(a,n);  //first sort from 0 to 100 the 0 to 200 and so on upto n  // mergeSort(a,0,n);  double mergeT = mergeCalc(a, n);  double quickT = qC(b, n);  printf("Time taken by Merge Sort: %f\n", mergeT);  printf("Time taken by Quick Sort: %f\n", quickT);  //printArr(a,n);  return 0;  } |
| CONCLUSION | Successfully studied merge and quicksort in C. |