|  |  |
| --- | --- |
| **NAME:** | Akshaya Karande |
| **UID:** | 2021300056 |
| **SUBJECT** | Design and Analysis of Algorithm |
| **EXPERIMENT NO:** | 2 (A) |
| **AIM:** | Experiment based on divide and conquer approach-Quick Sort and Merge Sort |
| **Algorithm** |  |
| **PROGRAM:** | #include <bits/stdc++.h>  #include <chrono>  #include <fstream>  using namespace std;  void swap(int \*a, int \*b)  {  int t = \*a;  \*b= \*a;  \*a = t;  }  int partition(int a[],int l,int h)  {  int pivot = a[l];  int i=l;  int j=h;  while(i<j)  {  while(a[i]<=pivot) i++;  while(a[j]>pivot) j--;  if(i<j) swap(a[i],a[j]);  }  swap(a[j],a[l]);  return j;  }  void quicksort(int a[],int l, int h)  {  if(l<h)  {  int pivot = partition(a,l,h);  quicksort(a,l,pivot-1);  quicksort(a,pivot+1,h);  }  }  void merge(int arr[], int l, int m, int r)  {  int n1 = m - l + 1;  int n2 = r - m;      int L[n1], R[n2];      for (int i = 0; i < n1; i++)  L[i] = arr[l + i];  for (int j = 0; j < n2; j++)  R[j] = arr[m + 1 + j];    int i = 0;      int j = 0;      int k = l;    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){  return;  }  int m = (l+r-1)/2;  mergeSort(arr,l,m);  mergeSort(arr,m+1,r);  merge(arr,l,m,r);  }  void printArr(int arr[],int n)  {  for(int i=0;i<n;i++) cout<<arr[i]<<" ";  }  int main()  {  int n=100000;  int arr[n];  ofstream outdata;  // outdata.open("output.csv");    //random values filling  for(int i=0;i<n;i++)  {  arr[i] = rand() % n;  // outdata<<arr[i]<<", ";  // cout<<arr[i]<<", ";  }  int copyarr[n];  copy(arr,arr+n,copyarr);  // for(int i=51000;i<=51005;i++)  // {  // cout<<arr[i]<<"\n";  // }  for(int i=10000;i<n;i+=10000){  for(int j=i;j<i+10;j++){  cout<<arr[j]<< " ";  }  cout<<endl;  }  for(int i=100;i<=n;i+=100)  {  //Insertion Sort  auto start = chrono::high\_resolution\_clock::now();  mergeSort(arr,0,i-1);  auto end = chrono::high\_resolution\_clock::now();  // cout<<"\n"<<i/100<<","<<chrono::duration\_cast<chrono::microseconds>(end - start).count()<<",";  //Selection Sort  start = chrono::high\_resolution\_clock::now();  quicksort(copyarr,0,i-1);  end = chrono::high\_resolution\_clock::now();  // cout<<chrono::duration\_cast<chrono::microseconds>(end - start).count();  }  } |
| **Conclusion**    **Conclusion:**  **In conclusion, both merge sort and selection sort are important sorting algorithms that can be used to arrange a list of elements in ascending or descending order.**  **Merge sort is a divide-and-conquer algorithm that works by recursively breaking down a list into smaller sublists, sorting those sublists, and then merging them back together. It has a time complexity of O(n log n) and is very efficient for sorting large lists.**  **On the other hand, selection sort is a simple comparison-based sorting algorithm that works by selecting the minimum element from the unsorted part of the list and placing it at the beginning. It has a time complexity of O(n^2) and is not very efficient for large lists.**  **In terms of performance, merge sort outperforms selection sort in most cases, especially for large lists. However, selection sort can be useful for small lists or for situations where memory usage is a concern.**  **Overall, the choice between merge sort and selection sort depends on the size of the list to be sorted, the available memory, and the desired performance.** | |