su



Algorithms and Advanced Programming

Student Giordano Bruno Passos de Jesus

**Part 1: Sorting and Searching: Algorithm Analysis**

**1.**

Merge sort has a time complexity of O(n log n) which is far better than the time complexity of bubble sort that is O(n2) hence we use merge sort in the program

public void sort(int left, int right) {*//method to break the array in 2 parts for merge sort* if (left < right) {  
 int middle = left + (right - left) / 2; *//determine the middle index for dividing* sort(left, middle);  
 sort(middle + 1, right);  
 merge(left, middle, right);  
 }  
}  
public void merge(int left, int middle, int right) {*//merging all the divided pieces to form the sorted array* int sizeL = middle - left + 1;  
 int sizeR = right - middle;  
 Student tempL[] = new Student[sizeL];  
 Student tempR[] = new Student[sizeR];  
 for (int i = 0; i < sizeL; i++) {  
 tempL[i] = stude[left + i];  
 }  
 for (int i = 0; i < sizeR; i++) {  
 tempR[i] = stude[middle + 1 + i];  
 }  
  
 int i = 0;  
 int j = 0;  
 int k = left;  
 while (i < sizeL && j < sizeR) {*//loop to put the smallest element in left part of the array* if (tempL[i].getStudentName().compareTo(tempR[j].getStudentName()) <= 0) {  
 stude[k] = tempL[i];  
 i++;  
 } else {  
 stude[k] = tempR[j];  
 j++;  
 }  
 k++;  
 }  
 while (i < sizeL) {  
 stude[k] = tempL[i];  
 i++;  
 k++;  
 }  
 while (j < sizeR) {  
 stude[k] = tempR[j];  
 j++;  
 k++;  
 }  
}

System.*out*.println("how many students?");  
int size = scan.nextInt();  
Process obj1 = new Process(size);*//object for 10 students*obj1.storeData();  
System.*out*.println(obj1.invalid +" invalid data");  
double start = System.*nanoTime*();*//recording the current time in nanoseconds*obj1.sort(0, obj1.index - 1);*//sorting the array*double end = System.*nanoTime*();*//recording the current time*double time = end - start;*//calculating the time taken to sort*System.*out*.println("the process took: " + time/1000 + " ms");*//displaying the time taken to sort in milliseconds*System.*out*.println("how many students?");  
size = scan.nextInt();  
Process obj2 = new Process(size);*//object for 100 students*obj2.storeData();  
System.*out*.println(obj2.invalid +" invalid data");  
start = System.*nanoTime*();  
obj2.sort(0, obj1.index - 1);  
end = System.*nanoTime*();  
time = end - start;  
System.*out*.println("the process took: " + time/1000 + " ms");  
  
System.*out*.println("how many students?");  
size = scan.nextInt();  
Process obj3 = new Process(size); *//object for 1000 students*obj3.storeData();  
System.*out*.println(obj3.invalid +" invalid data");  
start = System.*nanoTime*();  
obj3.sort(0, obj1.index - 1);  
end = System.*nanoTime*();  
time = end - start;  
System.*out*.println("the process took: " + time/1000 + " ms");  
  
System.*out*.println("how many students?");  
size = scan.nextInt();  
Process obj4 = new Process(size);*//object for 10000 students*obj4.storeData();  
System.*out*.println(obj4.invalid +" invalid data");  
start = System.*nanoTime*();  
obj4.sort(0, obj1.index - 1);  
end = System.*nanoTime*();  
time = end - start;  
System.*out*.println("the process took: " + time/1000 + " ms");

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input size | 10 students | 1\\00 students | 1000 students | 10000 students |
| Merge Sort | 49.563ms | 40.439ms | 42.224ms | 66.002ms |

Text

Description automatically generated

Chart, line chart

Description automatically generated

The time complexity of the binary search is O(log n) which is far better than the time complexity of linear search that is O(n). Hence the search is done using binary search

public void search(int id) {*//method to binary search* int left = 0;  
 int right = size - 1;  
  
  
 while (left <= right) {*//loop to find the middle element is the element to be searched or not* int middle = left + (right - left) / 2;*//determine the middle value* if (studeDuplicate[middle].getStudentCode() == id) {*//checking if the item in the middle is the required item or not* System.*out*.println("item is found\n" + studeDuplicate[middle].toString());  
 return;  
 } else if (studeDuplicate[middle].getStudentCode() < id) {*//searching the right part of the middle index* left = middle + 1;  
 } else {*//searching the left part of the middle index* right = middle - 1;  
 }  
 }  
 System.*out*.println("item not found");  
}

**Part 2: Defensive Programming and Exception Handling**

public void setModeOfTransport(String modeOfTransport) {  
 Scanner scan = new Scanner(System.*in*);  
 try{  
 if (modeOfTransport.equalsIgnoreCase("Bus")||modeOfTransport.equalsIgnoreCase("Train")||modeOfTransport.equalsIgnoreCase("Car")||modeOfTransport.equalsIgnoreCase("Motor Bike")||modeOfTransport.equalsIgnoreCase("Bike")||modeOfTransport.equalsIgnoreCase("E-scooter")||modeOfTransport.equalsIgnoreCase("Walk")||modeOfTransport.equalsIgnoreCase("E-scooter")){  
 ModeOfTransport = modeOfTransport;  
 }else {  
 throw new TransportException();  
 }  
 }catch (TransportException i) {  
 System.*out*.println("‘Unsupported Mode of Transport. Please correct the transportation mode by choosing from (Bus, Train, Car, Motor Bike, Bike, E-scooter, Walk)’");  
 System.*out*.println("Provide a valid Mode of Transportation");  
 ModeOfTransport = scan.next();  
 }  
}

public boolean validate(int distance) {*//method to validate distance* try {*//try block for distance validation* if (distance < 1 || distance > 500) {*//checking if the distance is valid* throw new DistanceException();*//throwing distance exception error if the distance is inappropriate* }  
 } catch (DistanceException e) {*//catch block in reference to the try block* invalid++;*//calculating the invalid number of input* return false;  
 }  
 return true;  
}