10/02/2021

**Experiment No:22**

**QUICK SORT AND MERGE SORT**

**AIM:**

Create a text file containing the name, height, weight of the students in a class. Perform Quick sort and Merge sort on this data and store the resultant data in two separate files. Also write the time taken by the two sorting methods into the respective files. Eg: Sony Mathew 5.5 60 Arun Sajeev 5.7 58 Rajesh Kumar 6.1 70

**DATA STRUCTURES USED:**

Arrays

**ALGORITHM:**

Algorithm Partition(A, p, r)

START

1. x = A[r]

2. i = p-1

3. for j = p to r

4. if (A[j] <= x)

5. i = i+1

6. if (i != j)

7. swap A[i] and A[j]

8. endif

9. endif

10. endfor

11. if (r != i+1)

12. swap A[i+1] and A[r]

13. endif

14. return i+1

STOP

Algorithm QuickSort(A, p, r)

START

1. if (p < r)

2. q = Partition(A, p ,r)

3. QuickSort(A, p, q-1)

4. QuickSort(A, q+1, r)

5. endif

STOP

Algorithm Merge(A, p, q, r)

START

1. n1 = q - p + 1

2. n2 = r - q

3. Declare L[n1], R[n2]

4. for i = 0 till n1

5. L[i] = A[p+i]

6. endfor

7. for j = 0 till n2

8. R[j] = A[q+j+1]

9. endfor

10. i = 0, j = 0,L[n1+1]=R[n2+1]=∞

11. for k = p to r

12. if (L[i] <= R[j])

13. A[k] = L[i]

14. i = i+1

15. else

20. A[k] = R[j]

21. j = j+1

25. endif

27. endfor

STOP

Algorithm MergeSort(A, p, r)

START

1. if (p < r)

2. q = floor((p+r)/2)

3. MergeSort(A, p, q)

4. MergeSort(A, q+1, r)

5. Merge(A, p, q, r)

6. endif

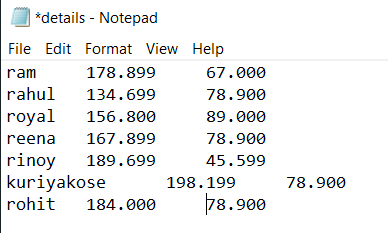
STOP

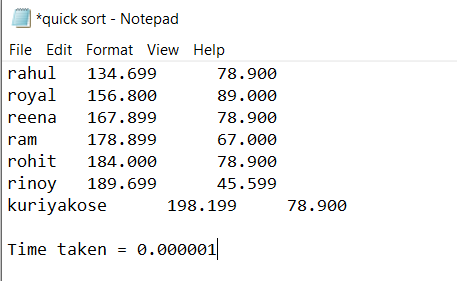
**PROGRAM:**

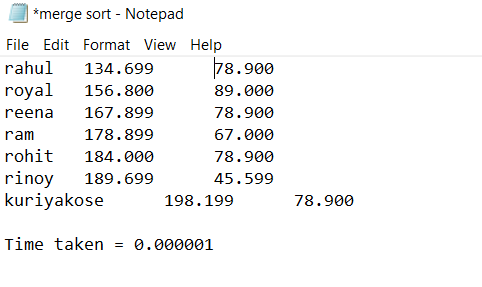
#include<stdio.h>  
#include<stdlib.h>  
#include<string.h>  
#include<time.h>  
#include<math.h>  
  
struct student  
{  
 char name[20];  
 float height;  
 float weight;  
};  
int partition(struct student s[], int p, int r){  
 struct student temp;  
 float x = s[r].height;  
 int i = p-1;  
 for(int j = p; j < r; j++){  
 if(s[j].height <= x){  
 i=i+1;  
 temp = s[i];  
 s[i] = s[j];  
 s[j] = temp;  
 }  
 }  
 temp = s[i+1];  
 s[i+1] = s[r];  
 s[r] = temp;  
 return i+1;  
}  
void quick\_sort(struct student s[], int p, int r)  
{  
 if(p < r)  
 {  
 int q = partition(s, p ,r);  
 quick\_sort(s, p, q-1);  
 quick\_sort(s, q+1, r);  
 }  
}  
void merge(struct student s[], int p, int q, int r){

int n1 = q - p + 1;  
 int n2 = r - q;  
 int i,j;  
 struct student L[n1], R[n2];  
 for( i = 0; i <n1; i++){  
 L[i] = s[p+i];  
 }  
 for(j = 0; j <n2;j++){  
 R[j] = s[q+j+1];  
 }  
 i = 0, j = 0;  
 int k;  
 for(k = p; k <= r; k++){  
 if(L[i].height <= R[j].height){  
 s[k] = L[i];  
 i=i+1;  
 if(i == n1){  
 k++;  
 break;  
 }  
 }  
 else{  
 s[k] = R[j];  
 j=j+1;  
 if(j == n2){  
 k++;  
 break;  
 }  
  
 }  
 }  
 while(i < n1)  
 {  
 s[k] = L[i];  
 i++;  
 k++;  
 }  
 while(j < n2)  
 {  
 s[k] = R[j];  
 j++;  
 k++;  
 }  
}  
  
void merge\_sort(struct student s[], int p, int r)  
{  
 if(p < r)  
 {  
 int q = floor((p+r)/2);  
 merge\_sort(s, p, q);  
 merge\_sort(s, q+1, r);  
 merge(s, p, q, r);  
 }  
}  
void main()  
{  
 int n,i;  
 char c,name[50];  
 float height, weight;  
  
 printf("Enter the Number of Students : ");  
 scanf("%d", &n);  
  
 FILE \*fp1 = fopen("details.txt", "w");  
 FILE \*fp2 = fopen("details.txt", "r");  
 FILE \*fp3 = fopen("quick sort.txt", "w");  
 FILE \*fp4 = fopen("merge sort.txt", "w");  
 struct student s1[50],s2[50];  
 for( i=0; i<n; i++){  
 printf("\nEnter the Details :\n");  
 printf(" Name : ");  
 scanf("%c", &c);  
 gets(name);  
 printf(" Height :");  
 scanf("%f", &height);  
 printf(" Weight : ");  
 scanf("%f", &weight);  
 fprintf(fp1,"%s\t%.3f\t%.3f\n",name,height,weight);  
 }  
 fclose(fp1);  
  
 for( i = 0; i <n; i++){  
 fscanf(fp2,"%s\t%f\t%f\n",s1[i].name, &s1[i].height, &s1[i].weight);  
 strcpy(s2[i].name,s1[i].name);  
 s2[i].height = s1[i].height;  
 s2[i].weight = s1[i].weight;  
 }  
 fclose(fp2);  
  
  
 clock\_t start ,stop;  
 start = clock();  
 quick\_sort(s1, 0, n-1);  
 stop= clock() ;  
 for(i = 0; i < n; i++){  
 fprintf(fp3,"%s\t%.3f\t%.3f\n",, s1[i].name, s1[i].height, s1[i].weight);  
 }  
 fprintf(fp3, "\nTime taken = %f", (double) (start-stop) / CLOCKS\_PER\_SEC);  
 fclose(fp3);  
  
 start = clock();  
 merge\_sort(s2, 0, n-1);  
 stop= clock();  
 for(i = 0; i < n; i++){  
 fprintf(fp4,"%s\t%.3f\t%.3f\n",, s2[i].name, s2[i].height, s2[i].weight);  
 }  
 fprintf(fp4, "\nTime taken = %f", (double) (start-stop) / CLOCKS\_PER\_SEC);  
 fclose(fp4);  
  
}

**OUTPUT:**







**RESULT:**

Quick sort and Merge sort were done on file containing data.

10/02/2021

**Experiment No:23**

**HEAP SORT**

**AIM:**

Write a program to sort a set of numbers using Heap sort and find a particular number from the sorted set using Binary Search.

**DATA STRUCTURES USED:**

Arrays

**ALGORITHM:**

Algorithm CreateHeap(A, n)

START

1. i = 0

2. while (i < n)

3. j = i

4. while (j > 1)

5. if (A[j] > A[(j-1)/2])

6. swap A[j] and A[(j-1)/2]

7. j = j/2

8. else

9. j=1

10. endif

11 endwhile

12.. i = i+1

13. endwhile

STOP

Algorithm RemoveMax(A, i)

START

1. swap A[i] and A[1]

STOP

Algorithm RebuildHeap(A, i)

START

1. if (i =1)

2. exit

3. endif

4. j = 1,flag=TRUE

5. while(flag = TRUE)

6. leftchild = 2 \* j

7. rightchild = 2 \*j + 1

8. if (rightchild <= i)

9. if(A[j] <= A[leftchild] && A[leftchild] >= A[rc])

10. swap A[j] and A[leftchild]

11. j = leftchild

12. else if (A[j] <= A[rightchild] && A[rightchild] >= A[leftchild])

13. swap A[j] and A[rightchild]

14. j = rightchild

15. else

16. flag = FALSE

17. else if (leftchild <= i)

18. if (A[j] <= A[leftchild])

19. swap A[j] and A[leftchild]

20. j=leftchild

21. else

22. break

23. else

24. flag = FALSE

25. endif

26. endwhile

STOP

Algorithm HeapSort(A, n)

START

1. CreateHeap(A, n)

2. for i = n-1 down till 2

3. RemoveMax(A, i)

4. RebuildHeap(A, i-1)

5. endfor

STOP

Algorithm BinarySearch(A, num, l, r)

START

1. while (first <= last)

2. middle = (first+last) / 2

3. if (A[middle] == num)

4. return middle

5. else if (A[middle] < num)

6. first = middle+1

7. else

8. last = middle - 1

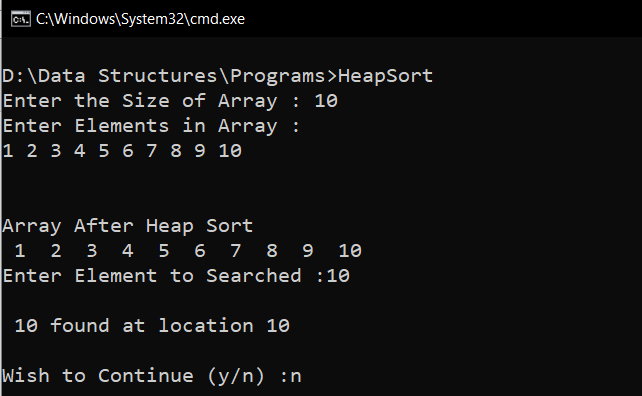
9. endif

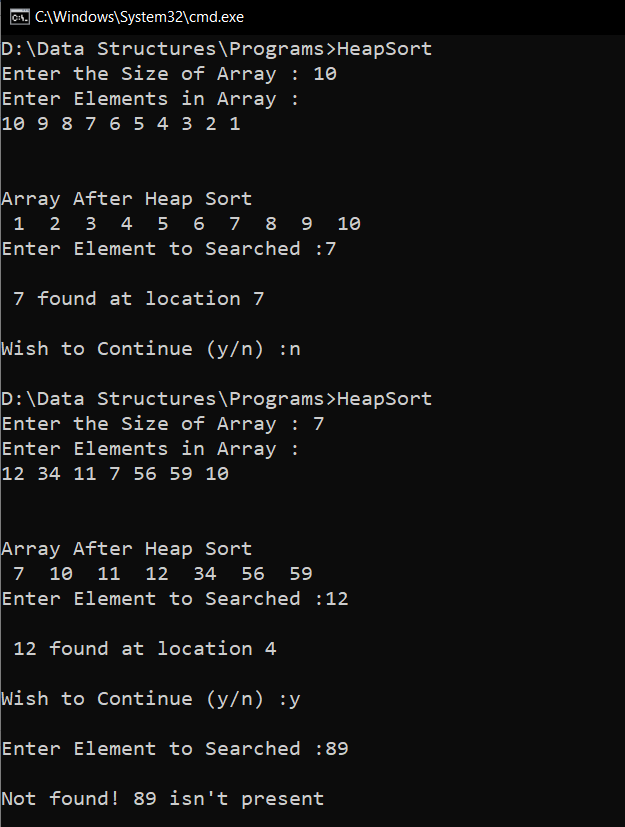
STOP

**PROGRAM:**

#include<stdio.h>  
void create\_heap(int A[],int B[],int n){  
 int i=1;  
 while(i<=n){  
 int x = A[i];  
 B[i] = x;  
 int j = i;  
 while(j>1){  
 if(B[j]>B[j/2]){  
 int temp = B[j];  
 B[j] = B[j/2];  
 B[j/2] = temp;  
 j = j/2;  
 }else{  
 j = 1;  
 }  
 }  
 i++;  
 }  
}  
  
void remove\_max(int B[],int i){  
 int temp = B[i];  
 B[i] = B[1];  
 B[1] = temp;  
}  
  
void rebuild\_heap(int B[],int i){  
 if(i!=1){  
 int j=1;  
 int flag = 1;  
 int temp;  
 while(flag==1){  
 int leftchild = 2\*j;  
 int rightchild = 2\*j+1;  
 if(rightchild<=i){  
 if((B[j]<=B[leftchild])&&(B[rightchild]<=B[leftchild])){  
 temp = B[j];  
 B[j] = B[leftchild];  
 B[leftchild] = temp;  
 j = leftchild;  
 }else if((B[j]<=B[rightchild])&&(B[rightchild]>=B[leftchild])){  
 temp = B[j];  
 B[j] = B[rightchild];  
 B[rightchild] = temp;  
 j = rightchild;  
 }else{  
 flag = 0;  
 }  
 }else if(leftchild<=i){  
 if(B[j]<=B[leftchild]){  
 temp = B[j];  
 B[j] = B[leftchild];  
 B[leftchild] = temp;  
 j = leftchild;  
 }else{  
 flag = 0;  
 }  
 }else{  
 flag=0;  
 }  
 }  
 }  
}  
  
void binary\_search(int B[],int n,int item){  
  
  
 int first = 1;  
 int last = n;  
 int middle = (first+last)/2;  
 while(first<=last) {  
 if(B[middle]<item){  
 first = middle + 1;  
 }  
 else if(B[middle] == item) {  
 printf("\n %d found at location %d \n", item, middle);  
 break;  
 }  
 else{  
 last = middle - 1;  
 }  
 middle = (first + last)/2;  
 }  
 if(first>last){  
 printf("\nNot found! %d isn't present \n", item);  
 }  
}  
  
void main(){  
 int A[100],B[100],n,i,item;  
 char ans;  
 printf("Enter the Size of Array : ");  
 scanf("%d",&n);  
 printf("Enter Elements in Array :\n");  
 for(i=1;i<=n;i++){  
 scanf("%d",&A[i]);  
 }  
 create\_heap(A,B,n);  
 for(i=n;i>1;i--){  
 remove\_max(B,i);  
 rebuild\_heap(B,i-1);  
 }  
 printf("\n\nArray After Heap Sort\n");  
 for(i=1;i<=n;i++){  
 printf(" %d ",B[i]);  
 }  
 do{  
 printf("\nEnter Element to Searched :");  
 scanf("%d", &item);  
 binary\_search(B,n,item);  
 printf("\nWish to Continue (y/n) :");  
 scanf("%c",&ans);  
 scanf("%c",&ans);  
 }while(ans=='y');  
}

**OUTPUT:**





**RESULT:**

Heap Sort was carried out in a set of data.