|  |
| --- |
|  |
| EXPERIMENT 4 |
|  |
|  |
| **Vansh Sukhija**  **12112021** |
|  |

|  |
| --- |
|  |

Ans 1

// Implement Linear search and Binary search on 1D array of Integers and also return the time complexity in all the cases i.e your worst case, average case, and best case.

#include<stdio.h>

*int* Lsearch(*int* *arr*[], *int* *n*, *int* *key*){

    for(*int* i=0; i<*n*; i++){

        if(*arr*[i]==*key*)

            return i;

    }

    return -1;

}

*void* sort(*int* *arr*[], *int* *n*){

    for(*int* i=*n*-1; i>0; i--){

        for(*int* j=0; j<i; j++){

            if(*arr*[j]>*arr*[j+1]){

*int* temp = *arr*[j];

*arr*[j] = *arr*[j+1];

*arr*[j+1] = temp;

            }

        }

    }

}

*int* Bsearch(*int* *arr*[], *int* *n*, *int* *key*){

    sort(*arr*, *n*);

*int* start=0, end=*n*;

    while(start<=end){

*int* mid = (start+end)/2;

        if(*arr*[mid]==*key*)

            return mid;

        else if(*arr*[mid]<*key*)

            start = mid+1;

        else

            end = mid-1;

    }

    return -1;

}

*void* main(){

*int* n, key;

    printf("Enter the number of elements in the array: ");

    scanf("%d", &n);

*int* arr[n];

    printf("Enter elements of array: ");

    for(*int* i=0; i<n; i++){

        scanf("%d", &arr[i]);

    }

    printf("Enter the key to be found: ");

    scanf("%d", &key);

*int* linear = Lsearch(arr, n, key);

    printf("Using linear search\n");

    if(linear>=0){

        printf("Key found at index: %d\nTime Complexity: ", linear);

        if(linear==n-1)

            printf("Worst\n");

        else if(linear==0)

            printf("Best\n");

        else

            printf("Average\n");

    }

    else

        printf("Key not found\nTime Complexity: Worst");

*int* binary = Bsearch(arr, n , key);

    printf("\nUsing binary search\n");

    if(binary>=0){

        printf("Key found at index: %d\nTime Complexity: ", binary);

        if(binary==(n-1)/2)

            printf("Best\n");

        else if(binary==0 || binary==n-1)

            printf("Worst\n");

        else

            printf("Average\n");

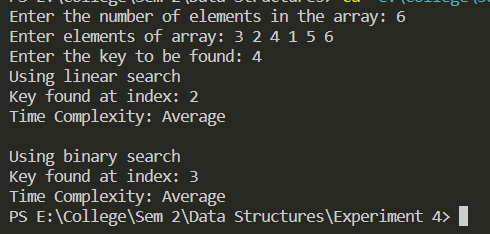
    }

    else

        printf("Key not found\nTime Complexity: Worst");

}

Output:



Ans 2:

// WAP to implement Bubble on 1D array of Faculty structures (contains faculty\_name, faculty\_ID, subject\_codes, class\_names) with key as faculty\_ID.

#include<stdio.h>

typedef *struct* faculty{

*char* name[20];

*int* id;

*char* code[7];

*char* class\_name[4];

}fac;

*void* sort(fac *arr*[], *int* *n*){

    for(*int* i=*n*-1; i>0; i--){

        for(*int* j=0; j<i; j++){

            if(*arr*[j].id>*arr*[j+1].id){

                fac temp = *arr*[j];

*arr*[j] = *arr*[j+1];

*arr*[j+1] = temp;

            }

        }

    }

}

*void* main(){

*int* n;

    printf("Enter the number of faculties: ");

    scanf("%d", &n);

    fac arr[n];

    for(*int* i=0; i<n; i++){

        fflush(stdin);

        printf("Enter name of faculty %d: ", i+1);

        gets(arr[i].name);

        printf("Enter ID: ");

        scanf("%d", &arr[i].id);

        fflush(stdin);

        printf("Enter Subject code: ");

        gets(arr[i].code);

        printf("Enter Class name: ");

        gets(arr[i].class\_name);

        printf("\n");

    }

    printf("Before sorting\n");

    for(*int* i=0; i<n; i++){

        puts(arr[i].name);

        printf("%d\n\n", arr[i].id);

    }

    sort(arr, n);

    printf("After sorting wrt faculty ids\n");

    for(*int* i=0; i<n; i++){

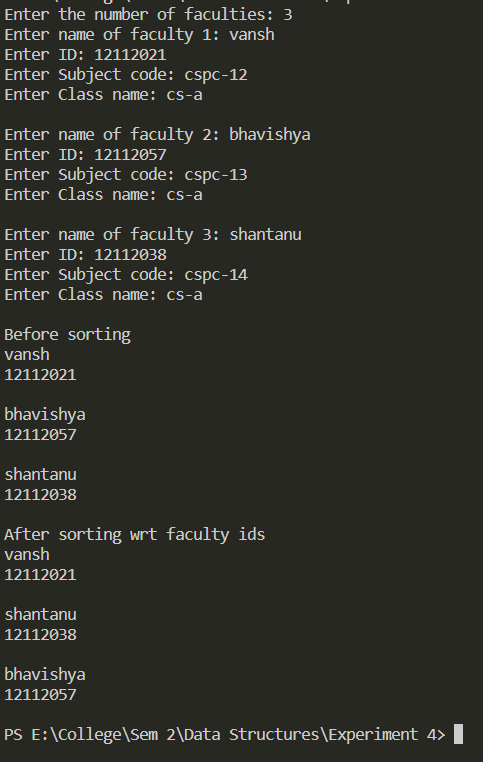
        puts(arr[i].name);

        printf("%d\n\n", arr[i].id);

    }

}

Output:



Ans 3:

// Implement Insertion and Selection sort on 1D array of integers and also return the time complexity in all the cases i.e your worst case, average case, and best case.

#include<stdio.h>

*void* Insertion(*int* *arr*[], *int* *n*){

    for(*int* i=0; i<*n*; i++){

*int* j=i+1;

        while(j>0 && *arr*[j]<*arr*[j-1]){

*int* temp = *arr*[j];

*arr*[j] = *arr*[j-1];

*arr*[j-1] = temp;

            j--;

        }

    }

}

*void* Selection(*int* *arr*[], *int* *n*){

    for(*int* i=0; i<*n*-1; i++){

        for(*int* j=i+1; j<*n*; j++){

            if(*arr*[i]>*arr*[j]){

*int* temp = *arr*[j];

*arr*[j] = *arr*[i];

*arr*[i] = temp;

            }

        }

    }

}

*void* main(){

*int* n;

    printf("Enter number of elements in array: ");

    scanf("%d", &n);

*int* arr1[n], arr2[n];

    printf("Enter elements in 1st array: ");

    for(*int* i=0; i<n; i++){

        scanf("%d", &arr1[i]);

    }

    Insertion(arr1, n);

    printf("Array 1 after Insertion sort: ");

    for(*int* i=0; i<n; i++){

        printf("%d ", arr1[i]);

    }

    printf("\n\n");

    printf("Enter elements in 2nd array: ");

    for(*int* i=0; i<n; i++){

        scanf("%d", &arr2[i]);

    }

    Selection(arr2, n);

    printf("Array 2 after Selection sort: ");

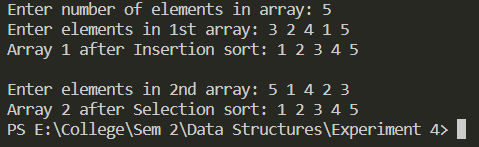
    for(*int* i=0; i<n; i++){

        printf("%d ", arr2[i]);

    }

}

Output:



Ans 4-a:

// Implement Quick Sort with duplicate numbers in the given array/elements

#include<stdio.h>

*void* swap(*int* \**a*, *int* \**b*){

*int* temp = \**a*;

    \**a* = \**b*;

    \**b* = temp;

}

*int* partition(*int* *arr*[], *int* *l*, *int* *h*){

*int* pivot = *arr*[*l*], i=*l*, j=*h*;

    while(i<j){

        do{

            i++;

        }while(*arr*[i]<=pivot);

        do{

            j--;

        }while(*arr*[j]>pivot);

        if(i<j)

            swap(&*arr*[i], &*arr*[j]);

    }

    swap(&*arr*[j], &*arr*[*l*]);

    return j;

}

*void* quicksort(*int* *arr*[], *int* *l*, *int* *h*){

    if(*l*<*h*){

*int* j = partition(*arr*, *l*, *h*);

        quicksort(*arr*, *l*, j);

        quicksort(*arr*, j+1, *h*);

    }

}

*void* main(){

*int* n;

    printf("Enter number of elements in the array: ");

    scanf("%d", &n);

*int* arr[n];

    printf("Enter elements of the array: ");

    for(*int* i=0; i<n; i++){

        scanf("%d", &arr[i]);

    }

    quicksort(arr, 0, n);

    printf("Sorted array: ");

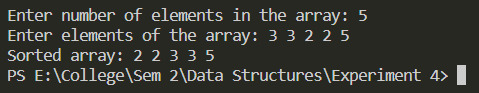
    for(*int* i=0; i<n; i++){

        printf("%d ", arr[i]);

    }

}

Output:



Ans 4-c:

// Implement Quick Sort using first/last/any random element as pivot

#include<stdio.h>

*void* swap(*int* \**a*, *int* \**b*){

*int* temp = \**a*;

    \**a* = \**b*;

    \**b* = temp;

}

*int* partition\_last(*int* *arr*[], *int* *l*, *int* *h*){

*l*--, *h*--;

*int* pivot = *arr*[*h*], i=*l*, j=*h*;

    while(i<j){

        do{

            i++;

        }while(*arr*[i]<pivot);

        do{

            j--;

        }while(*arr*[j]>=pivot);

        if(i<j)

            swap(&*arr*[i], &*arr*[j]);

    }

    swap(&*arr*[i], &*arr*[*h*]);

    return i;

}

*void* quicksort\_last(*int* *arr*[], *int* *l*, *int* *h*){

    if(*l*<*h*){

*int* j = partition\_last(*arr*, *l*, *h*);

        quicksort\_last(*arr*, *l*, j);

        quicksort\_last(*arr*, j+1, *h*);

    }

}

*void* main(){

*int* n;

    printf("Enter number of elements in the array: ");

    scanf("%d", &n);

*int* arr[n];

    printf("Enter elements of the array: ");

    for(*int* i=0; i<n; i++){

        scanf("%d", &arr[i]);

    }

    quicksort\_last(arr, 0, n);

    printf("Sorted array: ");

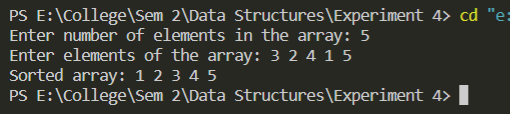
    for(*int* i=0; i<n; i++){

        printf("%d ", arr[i]);

    }

}

Output:



Ans 5:

// WAP to implement Merge Sort on 1D array of Student structures (contains student\_name, student\_roll\_no, total\_marks) with key as student\_roll\_no.

#include<stdio.h>

typedef *struct* student{

*char* name[20];

*int* rollno;

*int* marks;

}stud;

*void* merge(stud *st*[], *int* *l*, *int* *mid*, *int* *h*){

*int* n1 = *mid* - *l* + 1;

*int* n2 = *h* - *mid*;

    stud temp1[n1], temp2[n2];

    for(*int* i=0; i<n1; i++){

        temp1[i] = *st*[*l*+i];

    }

    for(*int* i=0; i<n2; i++){

        temp2[i] = *st*[*mid*+1+i];

    }

*int* i=0, j=0, k=*l*;

    while(i<n1 && j<n2){

        if(temp1[i].rollno<=temp2[j].rollno)

*st*[k++] = temp1[i++];

        else

*st*[k++] = temp2[j++];

    }

    while(i<n1){

*st*[k++] = temp1[i++];

    }

    while(j<n2){

*st*[k++] = temp2[j++];

    }

}

*void* mergesort(stud *st*[], *int* *l*, *int* *h*){

    if(*l*<*h*){

*int* mid = (*l*+*h*)/2;

        mergesort(*st*, *l*, mid);

        mergesort(*st*, mid+1, *h*);

        merge(*st*, *l*, mid, *h*);

    }

}

*void* main(){

*int* n;

    printf("Enter the number of students in class: ");

    scanf("%d", &n);

    stud st[n];

    for(*int* i=0; i<n; i++){

        fflush(stdin);

        printf("Enter name of student %d: ", i+1);

        gets(st[i].name);

        printf("Enter Roll no.: ");

        scanf("%d", &st[i].rollno);

        printf("Enter Total Marks: ");

        scanf("%d", &st[i].marks);

        printf("\n");

    }

    mergesort(st, 0, n);

    printf("---After sorting---\n");

    for(*int* i=0; i<n; i++){

        printf("\n");

        puts(st[i].name);

        printf("%d\n", st[i].rollno);

        printf("%d\n", st[i].marks);

    }

}

Output:

