CS19001: Programming and Data Structures Laboratory

Soumyajit Dey, Aritra Hazra;
CSE, IIT Kharagpur

http://cse.iitkgp.ac.in/~aritrah/course/lab/PDS/Autumn2018/CS19101_PDS-Lab_Autumn2018.html

29-Sep-2018

Programming Assignments

Complete and submit during lab

Assignment 1 [MinMax-Sort]

Write a C-program to perform MinMax-sort over an unordered n-element integer array to make the elements ascending-ordered.

Procedure

The working of the MinMax-sort is somewhat similar to that of selection sort. Here, the outer loop runs over (i, j) together, where i ranges from 0 up to $(\lfloor \frac{n}{2} \rfloor - 1)$ and j ranges from $(n-1)$ down to $\lceil \frac{n}{2} \rceil$. For given i, j, largest and smallest elements in the sub-array $A[i], A[i+1], \ldots, A[j-1], A[j]$ are found out (both together) and are swapped with the elements $A[j]$ and $A[i]$, respectively. Thus, during the first iteration of the outer loop $A[n-1]$ and $A[0]$ receives the largest and smallest element in the array, respectively; in the second iteration $A[n-2]$ and $A[1]$ receives the second-largest and second-smallest element, respectively and so on.

Example

$\{4,5,6,3,1,2\} \longmapsto$ after iteration 1 of outer loop $\longmapsto \{1,5,2,3,4,6\}$
$\{1,5,2,3,4,6\} \longmapsto$ after iteration 2 of outer loop $\longmapsto \{1,2,4,3,5,6\}$
$\{1,2,4,3,5,6\} \longmapsto$ after iteration 3 of outer loop $\longmapsto \{1,2,3,4,5,6\}$

Assignment 2 [Biparted-Ternary-Search]

Procedure

Consider a variation of binary search where the sorted array of size n is divided into two parts, but everytime by choosing the $n/3$-th element instead of the middle elements. The algorithm is as follows:

- Compare v (the searched element) with the $n/3$-th element
- If equal, v found – return
- If v is smaller, search first sub-array (0 to $n/3 - 1$)
- If v is greater, search middle sub-array ($n/3 + 1$ to $n - 1$)

Recursive-Function

Write a recursive C-function
int BiTernarySearch (int A[ ], int v, int low, int high)
which takes as parameters a sorted array A of integers, two indices *low* and *high* ($low \leq high$) in A and the element to be searched for v. The function returns the index, k ($low \leq k \leq high$), of A if v is found within the indices *low* and *high* (both included) of A, otherwise it returns -1.

Assignment 2 [Biparted-Ternary-Search]

Main-Program

Write a main C-function that

1. reads from user an integer *n* ($n \leq 100000$) and then takes from user *n* integers in an array (may be unordered);
2. reads another integer *x*, which is the element being searched;
3. sort the array elements in ascending order using previous **MinMax-Sort** program (Refer to *Assignment-1*);
4. checks whether *x* resides in the array or not, by using **BiTernarySearch** function;
5. prints the location/index where the element *x* resides in the array, otherwise print -1 in case it is not found.

Assignment 3 [Triparted-Ternary-Search]

Consider a variation of binary search where the sorted array of size n is divided into three parts instead of two parts by choosing the $n/3$-th and $2n/3$-th elements instead of only the middle elements. The algorithm is as follows:

- Compare v (the element being searched for) with the $n/3$-th element
- If equal, v found – return
- If v is smaller, search first sub-array (0 to $n/3 - 1$)
- If v is greater, compare with $2/3$-th element
- If equal, v found – return
- If v is smaller, search middle sub-array ($n/3 + 1$ to $2n/3 - 1$)
- If v is greater, search third sub-array ($2n/3 + 1$ to $n - 1$)

Assignment 3 [Triparted-Ternary-Search]

Recursive-Function

Write a recursive C-function
int TriTernarySearch (int A[ ], int v, int low, int high)
which takes as parameters a sorted array A of integers, two indices low and $high$ ($low \leq high$) in A and the element to be searched for v. The function returns the index, k ($low \leq k \leq high$), of A if v is found within the indices low and $high$ (both included) of A, otherwise it returns -1.

Main-Program

Write a main C-function that

1. reads from user an integer n ($n \leq 100000$) and then takes from user n integers in an array (must be in ascending order);
2. reads another integer x, which is the element being searched;
3. checks whether x resides in the array or not, by using **TriTernarySearch** function;
4. prints the location/index where the element x resides in the array, otherwise print -1 in case it is not found.

You do not have to sort the array. Just enter the numbers in sorted order directly from the keyboard.

Thank You