ACM40660/PH502 Assignment 2

ICHEC

Deadline: 1st December 2024 at 5:30pm

1 Objectives

- Understand memory management using dynamically allocated arrays in C/Fortran. Make use of pointers in C.
- Practice generating random numbers in C/Fortran.
- Implement sorting/searching algorithms. Make use of recursive functions.
- Learn how to do a simple benchmark using time measurement.
- Gain an understanding of how to evaluate time complexity of algorithms.

2 Questions

Download the code stub Q1_stub.c/f90 from Brightspace/Canvas.

1. Write a program that implements: generating a random array, sorting the array elements, searching for an element in the array and bechmarking the performance of the search for given parameters.

   Please follow the instructions below to complete the TODOs in the provided code stub.

   (a) Write a *randomarray(n, max)* function that returns a pointer to an array of size *n* integers, filled with random values between 0 and *max*.

   (b) Write a *printarray(n, arr[])* function that takes a pointer to an array of integers and prints its elements on screen.

   (c) Write a *quickSort(arr[], low, high)* function that implements quicksort algorithm that sorts the size *n* array of integers *arr* in ascending order where low and high are the starting and ending indices of the portion of the array to be sorted.

   QuickSort (`https://en.wikipedia.org/wiki/Quicksort`) is a divide-and-conquer sorting algorithm. It works by selecting an element called pivot from the array. Pick the last element as the pivot. Move elements less than or equal to the pivot to the left side, and elements greater than the pivot to the right. Place the pivot in its correct position. Then, recursively apply the same process to the left and right sub-arrays, continually dividing the array into smaller parts until each part contains only one or zero elements. Just to illustarte:

(24, 2, 10, 29, 16, 30, 18) pivot=18
Left part: (2, 10, 16) Right part: (24, 30, 29)
(2, 10, 16, 18, 24, 30, 29)

Left part:
(2, 10, 16) pivot=16
Left part: (2, 10) Right part: ()
(2, 10, 16)

Left part:
(2, 10) pivot=10
Left part: (2) Right part: ()
(2, 10)

Rigth part:
(24, 30, 29) pivot = 29
Left part: (24) Right part: (30)
(24, 29, 30)

(2, 10, 16, 18, 24, 29, 30) completely sorted array

(d) Write a *search(s, n, arr)* function that returns the smallest index of the size n array of integers *arr* containing the value s or -1 if it's not present. In order to do it, the function should assume that the array is sorted in ascending order. It will scan the list from the beginning until it finds the correct element or finds an element superior to s (the array being sorted, that means the element we're looking for is not present), or reaches the end of the array.

(e) Write a main program that asks the user for an array size and a maximum value. The program should then:

- generate the array using *randomarray* function,
- print it to the screen using *printarray* function,
- sort it using *quickSort* function and print it again using *printarray*,
- ask the user for a value to search and search using *search* function,
- print the index of the value if it is found.

(f) Write a *benchmark(n, max, s, mult)* function that will do the following:

- start a timer,
- loop for *mult* times,
  - for every 1000 cycles
    - generate an array through *randomarray()* using n and *max* as parameters
    - and sort the array through *quickSort()*,
  - search multiple times for the value s in the array using *search(s, n, arr)*,
- stop the timer,
- deallocate the array,

- calculate and return the elapsed time in seconds.

(g) Call your benchmark function in the main program with the following sets of parameters and print the elapsed time in seconds.
  - (2000, 10000, 10, 100000)
  - (2000, 10000, 5000, 100000)
  - (2000, 10000, 9000, 100000)

(h) What is your observations in the benchmark output in (g)?

(i) Can you explain the time complexity (worst-case and best-case) of the sort and search functions used in (c) and (d)?

Some general points:

- the main point is to get the program to do what the question asks (use either FORTRAN or C),
- make sure your code is syntactically correct (i.e. it compiles on sciprog),
- comment your code,
- make sure the work is your own and upload the completed code to BrightSpace/Canvas,
- make sure you upload the following:
  - Completed code for Question 1 as .c or .f90 file
  - Document for (h) and (i) as PDF/JPG/PNG format or can be included as comment lines at the end of your code.