CS213/293 Data Structure and Algorithms 2024 IITB India
Lab 1 - Introduction

Duration : 180 minutes **Total : 100 marks**

Instructions

1. The assignment is individual. Discussion is encouraged, but not recommended.
2. Plaigarism is strictly prohibited. We will run your code through MOSS and any cases detected will be reported to the DDAC. (The risk is yours to take.)
3. The lab is for 3 hours and no extra time will be given (unless everyone finds the lab hard).
4. These labs are graded, so write your code properly.
5. You cannot use ChatGPT/Gemini/Claude/Copilot/Siri/Llama to write code. As good as these people are, you have to learn to write on your own.

1. Binary Search

Consider the binary search algorithm presented in class. We are interested in determining the average execution time of the binary search. To accomplish this, let's conduct an experiment.

Suppose we have an array of size 1024, which contains distinct elements arranged in non-increasing order. We have already analyzed the running time when the element being searched for is not present in the array. Now, let's assume that we are only searching for elements that we know exist in the array.

Our goal is to experimentally calculate the average number of iterations required to search for all 1024 elements in the array.

In the lab files,

(a) Implement `BinarySearch` that can handle non-increasing array

(b) Harness `BinarySearch` such that we can compute avarage number of iterations

2. Performance of Arrays and Vectors

The key difference between arrays and vectors is that the size of an array is not flexible, while the size of a vector can be adjusted. For example, using `vector.push_back(a)` can extend the length of the vector by 1. However, this flexibility comes with an overhead cost for automatic size maintenance.

Our objective is to measure this cost by calculating the average access time (average cost of one read/write) to the container.

We will use the following simple test program:

(a) Declare the container.

(b) Assign container[i] = i for i ranging from 0 to N.

(c) Find the maximum value stored in the container and return the value.

The above program will have $2N$ accesses to the container (N writes and N reads). So

$$\text{average access time} = \frac{\text{runtime}}{2N}$$

If the container is an array, the size needs to be declared in advance. In the case of vectors, we will use `push_back` to insert elements and simulate the flexible size of vectors.

For arrays, write the following two versions of the test:

(a) Locally declared array.

(b) Globally declared array.

For vectors, write the following four versions of the test:

(a) Local vector.

(b) Local vector with reserve (refer to the C++ manual for the `reserve` function).

(c) Local vector with access to the vector using an iterator. For example, `for(int v : vector)`.

(d) Global vector (you may use the `static` keyword).

Run your experiments for $N = 2, 2\times 64, 2\times 64\times 64, 2\times 64\times 64\times 64, 2\times 64\times 64\times 64\times 64$ and compute the average runtime.

For measuring time use the `rdtsc.h` directory. It has the `ClockCounter` class, which measure time by counting clock of CPU.

Run your code for various compilation levels `-O0,-O1,-O2,-O3`.

3. Memory Allocation

Let's consider vectors again. When a vector is declared without specifying a size, it starts empty. However, there needs to be some initial allocation of space in anticipation of new elements being added. As we begin filling the vector, there will come a point when the allocated memory runs out. At this stage, the vector needs to allocate more memory.

Your task is to write code that detects at what size the new allocation occurs and determine the amount of extra memory being allocated.

Hint: Keep in mind that when a vector requires more memory, it not only needs additional space but also needs to be relocated to ensure continuous storage of the vector. This relocation process takes some time to execute.