# Laboratory 08B

CS-102

Spring 2022

# Searching Algorithms Intro.

- In Laboratory08A, we compared sorting algorithms. In Laboratory08B we will compare Search algorithms, namely the Linear search as well as the Binary search.
- We needed to do the sorting algorithms first because the Binary search algorithm depends upon the sequence (array or vector) being in sorted order to begin with.
- Now that we know one of the more efficient sorts for large sequences, we will use the QuickSort to put our sequence into ascending order, and then test the two different types of Searches to see how they compare in speed of the search.
- Our strategy will be to first sort a vector of 100,000,000 pseudo-random numbers, and then see how long it takes to find 99,999,999 – if it is there or not.
  - If it is there, it will be at the end of the list and will take the maximum amount of time for either a Linear Search or a Binary Search to find it.
  - If it is not there, the time it takes to not find it will be essentially the same.

# Searching Algorithms Strategy

- Attached is the Vector Version of the QuickSort program that we used in Lab08A. We will need to do two things to this program in order to make it compare Searches:
  1. We will need to add a search function, called lSearch() in the Linear search case and called bSearch() in the Binary search case, which will happen after the vector has been initially sorted.
  2. We will then need to move our chrono timer statements so that we are no longer measuring the amount of time it takes to sort the vector, but are measuring the amount of time it takes to search the Vector.
- You will need to take the linear search function that is given in program 8-1, linearSearch() which is written for arrays, not vectors, and convert it over to vector notation, and then do the same for the Binary search which is given in program 8-2 and is called binarySearch(). Both functions are attached.
- Be sure to add the appropriate prototypes to the beginning of your program.

# Searching Algorithms Part 1

- Using the programs given in the appendix of this lab, write two programs:
  - One that will use a Linear search to find the amount of time it takes to find (or not find) the number: 99,999,999. Call your program: ***YourName*-Lab08B-1a.cpp**
  - One that will use a Binary search to find the amount of time it takes to find (or not find) the number: 99,999,999. Call your program: ***YourName*-Lab08B-1b.cpp**
  - The only difference between the two programs will be the search function.
  - Remember Vectors are **passed by value** by default.
  - To minimize the search time, you will want to rewrite these to **pass the values by Reference**.
- Record the average amount of time it takes to do the Binary Search as well as the Linear search.
  - Run the test at least 5 times each.
- Save your results in a file called: ***YourName*-Lab08B-1.txt** .
- If you are doing this Lab synchronously, then demonstrate your results.
- If your are doing this Lab asynchronously, then submit your files to Canvas.

# Searching Algorithms Part 2

- Now, using your computer, and the program given to you in the appendix, record the amount of time it takes to sort N = 100,000,000 integers.
  - We did do this sort in the previous Lab. I ask you to do it again in case you have changed computers or have forgotten your previous result.
  - Note, you will need to move the chrono clock back to where it originally was in order to measure the time of the sort rather than the time of the search.
- Given the amount of time it takes to sort N integers, based on the data that you have accumulated, how many times would you need to search a database of N numbers before you would start saving time by having put the database into sorted order to begin with?
  - You may assume that the value that you are searching for, in an unsorted list will be found, on the average, after you have searched half-way through the list.
  - Report the ratio of the time it takes to sort the list of numbers to the average length of time it takes to find number you are searching for.

# Searching Algorithms Part 2

- Save your results in a file called: ***YourName*-Lab08B-2.txt** .
- If  you are doing this Lab synchronously, then demonstrate your results to the instructor.
- If you are doing this Lab asynchronously, then submit your file to Canvas.

# Vector QuickSort Algorithm I

```
// This program will sort MAX_NUM random numbers, using a QuickSort, into ascending order, & show the time of execution.
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <random>
#include <vector>
#include <chrono>
using namespace std;
void quick_sort(vector<int> &, int, int);
int partition(vector<int> &, int, int);
int bigRand(int, int);
void showStats(vector<int> &, int);
const int MAX_NUM =   100000000;
const int MAX_VALUE = 100000000;
const int MIN_VALUE = 0;
int main()
{
   vector<int>a(MAX_NUM);
   int elapsed_msecs, i;
   unsigned seed = time(0);
   srand(seed);
   for(i=0;i<MAX_NUM;i++)
        a[i]=bigRand(MAX_VALUE, MIN_VALUE);
   cout << "The clock has now started:" << endl << endl;
   auto started = chrono::high_resolution_clock::now();                    // Starts the clock
   quick_sort(a,0,MAX_NUM-1);
   auto done = chrono::high_resolution_clock::now();                       // Stops the clock
   elapsed_msecs = chrono::duration_cast<chrono::milliseconds>(done-started).count();
   cout << "Elapsed time = " << elapsed_msecs << " milliseconds." << endl;
   showStats(a,MAX_NUM);
   cin.get();
   return 0;
}
```

```
int bigRand(int upper, int lower)
{
   const int twoToFifteenth = 32768;
   int rand_num, rand_num1, rand_num2, randnum;
   int top = 32768;
   rand_num1 = 1 + rand() ;//% top;  // Check this out
   rand_num2 = rand() ;//% top;
   rand_num = twoToFifteenth*rand_num1 + rand_num2;
   randnum = rand_num%(upper - lower) + lower;
   return randnum;
}

void quick_sort(vector<int> & a,int k,int u)
{
   int j;
   if(k<u)
   {
      j=partition(a,k,u);
      quick_sort(a,k,j-1);
      quick_sort(a,j+1,u);
   }
}
```

# Vector QuickSort Algorithm II

```
int partition(vector<int> & a,int k,int u)
{
   int v,i,j,temp;
   v=a[k];
   i=k;
   j=u+1;
   do
   {
      do
            i++;
      while(a[i]<v&&i<=u);
      do
            j--;
      while(v<a[j]);
      if(i<j)
      {
            temp=a[i];
            a[i]=a[j];
            a[j]=temp;
      }
   }while(i<j);
   a[k]=a[j];
   a[j]=v;
   return(j);
}
```

# Vector QuickSort Algorithm III

```
void showStats(vector<int> & bigV, int size)
{
   int i;
   int smallest = bigV[0];
   int largest = bigV[0];
   long long total = bigV[0];
   for (i=1; i < size; i++)
   {
      largest = largest > bigV[i] ? largest : bigV[i];
      smallest = smallest < bigV[i] ? smallest : bigV[i];
      total += bigV[i];
   }
   cout << "\n" << "Number of numbers: " << size << endl;
   cout << "\n" << "Mean of numbers: " << total/size << endl;
   cout << "\n" << "Median number: " << bigV[size/2] << endl;
   cout << "\n" << "Smallest number: " << smallest << endl;
   cout << "\n" << "Largest number: " << largest << endl;
}
```

# Vector QuickSort Algorithm IV

```
int searchList(const int list[], int numElems, int value)
{
   int index = 0;                          // Used as a subscript to search array
   int position = -1;                      // To record position of search value
   bool found = false;                     // Flag to indicate if the value was found
   while(index < numElems && !found)
   {
        if (list[index] == value)          // If the value is found
        {
             found = true;                 // Set the flag
              position = index;            // Record the value's subscript
        }
        index++;                           // Go to next element
   }
   return position;                        //  Return the position, or -1
}
```

# Linear Search Function (Array version) Extracted from Program 8-1

```
int binarySearch(const int array[], int numElems, int value)
{
   int first = 0,                  // First array element
       last = numElems - 1,        // Last array element
       middle,                     // Mid point of search
       position = -1;              // Position of search value
   bool found = false;             // Flag
   while (!found && first <= last)
   {
            middle = (first + last) / 2;          // Calculate mid point
            if (array[middle] == value)     // If value is found at mid
            {
                     found = true;
                     position = middle;
             }
             else if (array[middle] > value)      // If value is in lower half
                     last = middle - 1;
             else
                     first = middle + 1;          // If value is in upper half
   }
   return position;
}
```

# Binary Search Function (Array version) Extracted from Program 8-2

```
// Demetris Leadership Center (DLC) product lookup program
// This program allows the user to enter a product number
// and then displays the title, description, and price of
// that product.
#include <iostream>
#include <string>
using namespace std;

const int NUM_PRODS = 9;              // The number of products produced
const int MIN_PRODNUM = 914;          // The lowest product number
const int MAX_PRODNUM = 922;          // The highest product number

// Function prototypes
int getProdNum();
int binarySearch (const int [], int, int);
void displayProd(const string [], const string [], const double [], int);
```

# Program 8-3 page 1

```
int main()
{
        // Array of product IDs
        int id[NUM_PRODS] = {914, 915, 916, 917, 918, 919, 920, 921, 922};

        // Array of product titles
        string title[NUM_PRODS] =
                        { "Six Steps to Leadership",
                          "Six Steps to Leadership",
                          "The Road to Excellence",
                          "Seven Lessons of Quality",
                          "Seven Lessons of Quality",
                          "Seven Lessons of Quality",
                          "Teams Are Made, Not Born",
                          "Leadership for the Future",
                          "Leadership for the Future"
                        };
```

# Program 8-3 page 2

```
// Array of product descriptions
string description[NUM_PRODS] =
                        { "Book", "Audio CD", "DVD",
                          "Book", "Audio CD", "DVD",
                          "Book", "Book", "Audio CD"
                        };

// Array of product prices
double prices[NUM_PRODS] = {12.95, 14.95, 18.95, 16.95, 21.95,
                                    31.95, 14.95, 14.95, 16.95};

int prodNum;        // To hold a product number
int index;          // To hold search results
char again;         // To hold a Y or N answer
```

# Program 8-3 page 3

```
	do
	{
		// Get the desired product number.
		prodNum = getProdNum();

		// Search for the product number.
		index = binarySearch(id, NUM_PRODS, prodNum);

		// Display the results of the search.
		if (index == -1)
			cout << "That product number was not found.\n";
		else
			displayProd(title, description, prices, index);

		// Does the user want to do this again?
		cout << "Would you like to look up another product? (y/n) ";
		cin >> again;
	} while (again == 'y' || again == 'Y');
	return 0;
}
```

# Program 8-3 page 4

```
//*************************************************
// Definition of getProdNum function             *
// The getProdNum function asks the user to enter a *
// product number. The input is validated, and when *
// a valid number is entered, it is returned.      *
//*************************************************
int getProdNum()
{
        int prodNum; // Product number

        cout << "Enter the item's product number: ";
        cin >> prodNum;
        // Validate input
        while (prodNum < MIN_PRODNUM || prodNum > MAX_PRODNUM)
        {
                cout << "Enter a number in the range of " << MIN_PRODNUM;
                cout <<" through " << MAX_PRODNUM << ".\n";
                cin >> prodNum;
        }
        return prodNum;
}
```

# Program 8-3 page 5

```
// Definition of binarySearch function. The binarySearch function performs a binary search on an    *
// integer array. array, which has a maximum of numElems elements, is searched for the number
// stored in value. If the number is found, its array subscript is returned. Otherwise, *
// -1 is returned indicating the value was not in the array.    *
int binarySearch(const int array[], int numElems, int value)
{
        int first = 0,                  // First array element
        last = numElems - 1,            // Last array element
        middle,                         // Midpoint of search
        position = -1;                  // Position of search value
        bool found = false;             // Flag
        while (!found && first <= last)
        {
                middle = (first + last) / 2;        // Calculate midpoint
                if (array[middle] == value)         // If value is found at mid
                {
                        found = true;
                        position = middle;
                }
                else if (array[middle] > value)     // If value is in lower half
                        last = middle - 1;
                else
                        first = middle + 1;         // If value is in upper half
        }
        return position;
}
```

# Program 8-3 page 6

```
//**************************************************************
// The displayProd function accepts three arrays and an int. *
// The arrays parameters are expected to hold the title,     *
// description, and prices arrays defined in main. The index *
// parameter holds a subscript. This function displays the   *
// information in each array contained at the subscript.     *
//**************************************************************

void displayProd(const string title[], const string desc[],
                 const double price[], int index)
{
                 cout << "Title: " << title[index] << endl;
                 cout << "Description: " << desc[index] << endl;
                 cout << "Price: $" << price[index] << endl;
}
```

# Program 8-3 page 7