**C++ Programming Name \_\_\_\_\_Tyler Hunt\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Arrays, Searching, and Sorting**

What are the *two* fundamental characteristics of arrays?

* Identical data types that are stored contiguously

What are the *two* consequences of those characteristics?

* Random access of the array
* Arrays are inflexible can’t change size

In Java, arrays are implemented as objects.

(a) true (b) false

In C++, arrays are implemented as objects.

(a) true (b) false

In Java, arrays are always passed to methods by

(a) value (b) pointer

(c) reference (d) token

In C++, arrays are always passed to functions by

(a) value (b) pointer

(c) reference (d) token

Arrays in Java have an integer instance variable called \_\_\_size\_\_\_\_ that holds the number of elements in the array.

What is the distinction between a *static* array and a *dynamic* array?

* Static - Stored on the stack initialized
* Dynamic - Initialized at run time and stored in the heap

What are the steps in creating static arrays in C++ and in Java?

* C++ Int\* a = {1,2,3,4};
* Java Int a[] = {12,3,4};

What are the steps in creating dynamic arrays in C++ and in Java?

* C++ Int\* a = new int[size];
* Java Int a[] = new int[size]

You wish to create a static array, named myArray, that will contain 1000 integers, and set each element in the array equal to zero.

Write the line(s) of Java and C++ code that accomplishes this.

* int \* n = new int[1000];
* int n[] = new int[1000]:

What if we wanted every element to be initialized to 1 instead?

for(int i = 0; i < 1000; i++){

n[i] = 1;

}

An array is declared as

int myArray [] = new int [4];

This array is a \_\_\_\_dynamic\_\_\_\_ array.

(a) static (b) dynamic

This array declaration is legal in

(a) C++ (b) Java

(c) both (d) neither

Another array is declared as

int myArray [] = {6, 0, 6, 0, 8, 4, 2};

This array is a \_static\_\_\_\_\_ array.

(a) static (b) dynamic

This array declaration is legal in

(a) C++ (b) Java

(c) both (d) neither

The index of the first element of an array is 1, and the index of the last element is equal to the number of elements in the array, minus one.

(a) true (b) false

Before any array operation, \_\_\_\_ determines whether or not the requested array index is in bounds.

(a) C++ (b) Java

(c) both (d) neither

Which of the following statements is true? Answer for C++, then for Java.

(a) An array cannot be sized dynamically

(b) An array can be sized dynamically, but cannot be resized without

instantiating it again.

(c) An array can be sized dynamically and can be resized without instantiating

it again.

(d) None of the above statements are true.

To multiply every element in an array named myArray by 4, use the statement

myArray \*= 4;

(a) true (b) false

If your answer to the previous question was “false,” write the C++ and Java code that accomplishes this task below.

myArray[i] \*=4;

An unsorted array can be quickly searched using binary search.

(a) true (b) false

If your answer to the above question was “false” (and I certainly hope that is was) write down the type of search you would use on an unsorted array.

Linear Search

You wish to create a two-dimensional array with NUM\_ROWS rows and NUM\_COLS columns. Write the code to accomplish this in C++ and in Java.

int cpp\_arr[][] = new int[ROWS][ NUM\_COLS];

int java\_arr[][] = new int[ROWS][ NUM\_COLS];

Now initialize that table so that every element contains the larger of its row and column index. For example, entry [2][3] would contain 3. Do this in C++ and in Java.

for(int i = 0; i<NUM\_ROWS;i++){

for(int j=0; j < NUM\_COLS; j++ ){

if(NUM\_ROWS > NUM\_COLS){

cpp\_arr[i][j] = NUM\_ROWS;

}else{

cpp\_arr[i][j] = NUM\_COLS;

}

}

}

Write code to implement sequential search in C++.

bool sequentialSearch(int searchItem, int \* theArray, int size){

for(int i = 0; i < size; i++){

if(theArray[i]== searchItem)

return true;

}

return false;

}

Write code to implement binary search in C++.

bool binarySearch( int searchItem, int \* theArray, int size){

int low = theArray[0];

int high = theArray[size-1];

if(low > high)return false;

int mid = (low + high) /2;

if(theArray[mid] < searchItem){

return binarySesarch(searchItem, theArray, low, mid - 1);

}else{

return binarySearch(searchItem, theArray, mid + 1, high);

}

}

How would you implement binary search using recursion?

See above.

Write code to exchange two items in an array, in both C++ and in Java.

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

int temp = \*a;

\*a = \*b;

\*b = temp;

}

the asymptotic performance of each of the three algorithms above.

How could we actually measure this?

Consider the following list of ten integers.

14 88 43 10 43 19 2 57 84 86

How many inversions are in this list?

45

When sorting this list, how many comparisons will be made by each of the three simple sorting algorithms?

exchange sort: 45

insertion sort: 17

selection sort: 45

When sorting this list, how many exchanges will be made by each of the three simple sorting algorithms?

exchange sort: 44

insertion sort: 9

selection sort: 9

A wood-chuck requires 20 seconds to chuck 60 chucks of wood, and the wood-chuck’s wood-chucking algorithm has temporal asymptotic performance **(*n*2). How much wood could this wood-chuck chuck (if a wood-chuck could chuck wood) in five minutes?

20 \* 3 = 180 min

180 \* 5 = 900 chunks of Wood

900 chunks of wood

A smarter woodchuck also requires 20 seconds to chuck 60 chucks of wood, but runs an algorithm with temporal asymptotic performance **(*n* lg *n*). How much time is required for the wood-chuck to chuck 5,000 chucks of wood?

How much wood could the first wood-chuck chuck in that amount of time?

For each of the three simple sorting algorithms, state the expected number of *comparisons* that will be made by…

Selection Insertion Exchange

Sort Sort Sort

a sorted list with 499500 257779 499500

length 1000:

a list of 1000 499500 257779 499500

random integers:

a reverse-sorted list: 499500 499451 499500

of 1000 integers

For each of the three simple sorting algorithms, state the expected number of

*exchanges* that will be made by…

Selection Insertion Exchange

Sort Sort Sort

a sorted list with

length 1000: 999 999 0

a list of 1000 999 999 257779

random integers:

a reverse-sorted list: 999 999 499451

of 1000 integers

Show the state of the following list of integers after three passes by insertion sort. Circle every number that is assured to be in its final sorted position.

10 45 23 7 64 88 43 10 43 19 2 57 84 26 45 99 14 74 89 2 99

10 23 45 7 64 88 43 10 43 19 2 57 84 26 45 99 14 74 89 2 99

7 10 23 45 64 88 43 10 43 19 2 57 84 26 45 99 14 74 89 2 99

7 10 23 45 64 88 43 10 43 19 2 57 84 26 45 99 14 74 89 2 99

7 10 23 45 64 88 43 10 43 19 2 57 84 26 45 99 14 74 89 2 99

7 10 23 43 45 64 88 10 43 19 2 57 84 26 45 99 14 74 89 2 99

7 10 10 23 43 45 64 88 43 19 2 57 84 26 45 99 14 74 89 2 99

7 10 10 23 43 43 45 64 88 19 2 57 84 26 45 99 14 74 89 2 99

7 10 10 19 23 43 43 45 64 88 2 57 84 26 45 99 14 74 89 2 99

2 7 10 10 19 23 43 43 45 64 88 57 84 26 45 99 14 74 89 2 99

2 7 10 10 19 23 43 43 45 57 64 88 84 26 45 99 14 74 89 2 99

2 7 10 10 19 23 43 43 45 57 64 84 88 26 45 99 14 74 89 2 99

2 7 10 10 19 23 26 43 43 45 57 64 84 88 45 99 14 74 89 2 99

2 7 10 10 19 23 26 43 43 45 45 57 64 84 88 99 14 74 89 2 99

2 7 10 10 19 23 26 43 43 45 45 57 64 84 88 99 14 74 89 2 99

2 7 10 10 14 19 23 26 43 43 45 45 57 64 84 88 99 74 89 2 99

2 7 10 10 14 19 23 26 43 43 45 45 57 64 74 84 88 99 89 2 99

2 7 10 10 14 19 23 26 43 43 45 45 57 64 74 84 88 89 99 2 99

2 2 7 10 10 14 19 23 26 43 43 45 45 57 64 74 84 88 89 99 99

2 2 7 10 10 14 19 23 26 43 43 45 45 57 64 74 84 88 89 99 99

2 2 7 10 10 14 19 23 26 43 43 45 45 57 64 74 84 88 89 99 99

Show the state of the following list of integers after three passes by selection sort. Circle

every number that is assured to be in its final sorted position.

10 45 23 7 64 88 43 10 43 19 2 57 84 26 45 99 14 74 89 2 99

2 45 23 7 64 88 43 10 43 19 10 57 84 26 45 99 14 74 89 2 99

2 2 23 7 64 88 43 10 43 19 10 57 84 26 45 99 14 74 89 45 99

2 2 7 23 64 88 43 10 43 19 10 57 84 26 45 99 14 74 89 45 99

2 2 7 10 64 88 43 23 43 19 10 57 84 26 45 99 14 74 89 45 99

2 2 7 10 10 88 43 23 43 19 64 57 84 26 45 99 14 74 89 45 99

2 2 7 10 10 14 43 23 43 19 64 57 84 26 45 99 88 74 89 45 99

2 2 7 10 10 14 19 23 43 43 64 57 84 26 45 99 88 74 89 45 99

2 2 7 10 10 14 19 23 43 43 64 57 84 26 45 99 88 74 89 45 99

2 2 7 10 10 14 19 23 26 43 64 57 84 43 45 99 88 74 89 45 99

2 2 7 10 10 14 19 23 26 43 64 57 84 43 45 99 88 74 89 45 99

2 2 7 10 10 14 19 23 26 43 43 57 84 64 45 99 88 74 89 45 99

2 2 7 10 10 14 19 23 26 43 43 45 84 64 57 99 88 74 89 45 99

2 2 7 10 10 14 19 23 26 43 43 45 45 64 57 99 88 74 89 84 99

2 2 7 10 10 14 19 23 26 43 43 45 45 57 64 99 88 74 89 84 99

2 2 7 10 10 14 19 23 26 43 43 45 45 57 64 99 88 74 89 84 99

2 2 7 10 10 14 19 23 26 43 43 45 45 57 64 74 88 99 89 84 99

2 2 7 10 10 14 19 23 26 43 43 45 45 57 64 74 84 99 89 88 99

2 2 7 10 10 14 19 23 26 43 43 45 45 57 64 74 84 88 89 99 99

2 2 7 10 10 14 19 23 26 43 43 45 45 57 64 74 84 88 89 99 99

2 2 7 10 10 14 19 23 26 43 43 45 45 57 64 74 84 88 89 99 99

2 2 7 10 10 14 19 23 26 43 43 45 45 57 64 74 84 88 89 99 99

Show the state of the following list of integers after three passes by exchange sort. Circle

every number that is assured to be in its final sorted position.

10 45 23 7 64 88 43 10 43 19 2 57 84 26 45 99 14 74 89 2 99

10 23 45 7 64 88 43 10 43 19 2 57 84 26 45 99 14 74 89 2 99

10 23 7 45 64 88 43 10 43 19 2 57 84 26 45 99 14 74 89 2 99

10 23 7 45 64 43 88 10 43 19 2 57 84 26 45 99 14 74 89 2 99

10 23 7 45 64 43 10 88 43 19 2 57 84 26 45 99 14 74 89 2 99

10 23 7 45 64 43 10 43 88 19 2 57 84 26 45 99 14 74 89 2 99

10 23 7 45 64 43 10 43 19 88 2 57 84 26 45 99 14 74 89 2 99

10 23 7 45 64 43 10 43 19 2 88 57 84 26 45 99 14 74 89 2 99

10 23 7 45 64 43 10 43 19 2 57 88 84 26 45 99 14 74 89 2 99

10 23 7 45 64 43 10 43 19 2 57 84 88 26 45 99 14 74 89 2 99

10 23 7 45 64 43 10 43 19 2 57 84 26 88 45 99 14 74 89 2 99

10 23 7 45 64 43 10 43 19 2 57 84 26 45 88 99 14 74 89 2 99

10 23 7 45 64 43 10 43 19 2 57 84 26 45 88 14 99 74 89 2 99

10 23 7 45 64 43 10 43 19 2 57 84 26 45 88 14 74 99 89 2 99

10 23 7 45 64 43 10 43 19 2 57 84 26 45 88 14 74 89 99 2 99

10 23 7 45 64 43 10 43 19 2 57 84 26 45 88 14 74 89 2 99 99

10 7 23 45 64 43 10 43 19 2 57 84 26 45 88 14 74 89 2 99 99

10 7 23 45 43 64 10 43 19 2 57 84 26 45 88 14 74 89 2 99 99

10 7 23 45 43 10 64 43 19 2 57 84 26 45 88 14 74 89 2 99 99

10 7 23 45 43 10 43 64 19 2 57 84 26 45 88 14 74 89 2 99 99

10 7 23 45 43 10 43 19 64 2 57 84 26 45 88 14 74 89 2 99 99

10 7 23 45 43 10 43 19 2 64 57 84 26 45 88 14 74 89 2 99 99

10 7 23 45 43 10 43 19 2 57 64 84 26 45 88 14 74 89 2 99 99

10 7 23 45 43 10 43 19 2 64 57 84 26 45 88 14 74 89 2 99 99

10 7 23 45 43 10 43 19 2 57 64 84 26 45 88 14 74 89 2 99 99………

2 2 7 10 10 14 19 23 26 43 43 45 45 57 64 74 84 88 89 99 99

# of comaprrisons: 210

# of exchanges: 79

Elapsed time: 0.573587 ms

Of the three “simple” sort algorithms, the one that performs the smallest number of comparisons for a randomly-ordered list is

(a) exchange sort (b) selection sort

(c) insertion sort (d) none of these are correct

Of the three “simple” sort algorithms, the one that performs the smallest number of exchanges for a randomly-ordered list is

(a) exchange sort (b) selection sort

(c) insertion sort (d) none of these are correct

Of the three “simple” sort algorithms, the one that has the best performance when sorting a nearly-ordered list is

(a) exchange sort (b) selection sort

(c) insertion sort (d) none of these are correct

All of our sorting algorithms have quadratic performance. Can this be improved? If so, to what extent? What is the best asymptotic performance that a sorting algorithm can have?

O(n)

------------------------------------------------------------------------------------------------------------

Class Notes:

Big O = shows how the time compares as the array grows

Sort Theory

Inversion – any pair of value in a sequence that are out of order

What is sorting? Make the inversions go away

**42 has 7 moves to the right**

**10 has 1 “”**

**18 has 4 “”**

**and so on**

**42 10 18 99 27 63 15 10 10 4**

10 element array takes 9 passes

Binary Search – O(Log n)

Sequential search - O(n)

**#compares #exchanges**

Exchange sort O(n^2) n(n-1)/2 #inversions

Insertion Sort Theta(n^2) inversion + n #inversions

Selection sort n(n-1)/2 < = n-1

Watch array boundries

**Metrics**

Random

Sorted

reversed