**CS 121 Week 12 Worksheet - Algorithm Analysis**

**Miscellaneous notes:**

* **Algorithm:** A method or set of rules used to solve a problem
* **Computational Problem:** A problem solved by an algorithm
* **Basic Step:** An operation in the algorithm that executes in a constant amount of time
* **Complexity of an Algorithm:** Number of basic steps required to execute an algorithm for an input of size N (where N equals the number of input values)
* Performance of an algorithm is usually noted by the amount of **time** and **space** they take to run
  + - **Time:** Time can be counted from the amount of milliseconds it takes to run (by using a timer) or the amount of (basic) steps it takes (in best, average, and worst-case scenarios). Mostly, we're concerned with the worst-case scenarios.
      * **Big-O Notation:** A way to write what the complexity of an algorithm is in its **worst case**. This is noted by saying *O(some complexity)*, as shown below:
        + describes an algorithm that will always execute in the same amount of steps, regardless of input data. This includes basic operations like adding and assignment.
        + describes an algorithm that cuts steps in half until its operation is complete. This includes algorithms like the **binary search**.
        + describes an algorithm whose performance grow in direct proportion to the input data set. This includes algorithms such as **sequential search**.
        + describes an algorithm whose performance grows to the square of the size of the input data set. This relates back to **bubble sort** and **selection sort** algorithms.
        + Although there are other types of complexities, the above four are most common.
        + Keep in mind that Big-O notation is generalized (e.g. )
    - **Space**: Space entails how much extra memory is required to run the algorithm (e.g. if it needs to make an entire copy of the list, add more variables, etc.).
      * All the algorithms we've talked about so far take up space (they don't need to create extra items proportional to the input data).
* **Search**: To locate a specific item in a list of values. At the moment, we're concerned with just two ways to search for a value in a list:
  + - **Sequential/Linear Search:** Start from the beginning, check each position sequentially until the value required is found. List can be in any order.
    - **Binary Search:** Check middle value until found. Requires list to be in order.
* **Sort:** To put a list of values (array) into a specific order. Typically, the values are put in alphabetical, ascending (least to greatest), or descending (greatest to least) order.
* There are many ways to sort an array. Here are two:
  + - **Bubble Sort:** Basic idea is to swap values through the list until they're in order.
    - **Selection Sort:** Have two regions (sorted and unsorted). Find minimum/maximum and swap with front index. Shift front index up one and repeat search, doing so until they're in order.
* **Psuedo-code example for Linear Search (edited from power point):**

*Set found to false*

*Set index to 0*

*while found is false and have not reached end of list*

*if current position at index is equal to value being searched for*

*set found to true*

*increment index*

* **Psuedo-code example for Binary Search (returns index where value is located; from PPT). Note that the below is for an ascending list (change the middle "Else If" to less than for descending):**

*Set first to 0*

*Set last to SIZE -1 (last subscript in array)*

*Set found to false*

*Set position to -1*

*While found is not true and first is less than or equal to the last*

*Set middle to the subscript halfway between first and last*

*If array[middle] equals desired value*

*Set found to true*

*Set position to middle*

*Else if array[middle] is greater than desired value*

*Set last to middle - 1*

*Else*

*Set first to middle + 1*

*End If*

*End While*

*Return position*

* **Psuedo-code example for Bubble Sort (shortened explanation from cplusplus.com):**

*For each element in the list (except for the last element)*

*if the element and the next element need swapping:*

*swap them*

*repeat if any swaps were made*

* **Explanation of Selection Sort (brief explanation from cplusplus.com):**

*Choose the smallest element of the unsorted items and place it at the end of the sorted part. Keep doing this until there are no more unsorted items to choose.  
  
The way to move an item from the unsorted to the sorted part is simple. Just swap the first item in the unsorted part with the item to move, and increase the size of the sorted part by one (to include the newly-sorted item).*

------------------------------------------------------------------------------------------------------------------------------------------**Algorithm Analysis Practice:**

1. Analyze the amount of steps the following code takes per function, including each function's Big-O notation.

#include <iostream>

using namespace std;

//Function prototypes

void init\_num\_list(int \* arr); //initializes an integer list (function

//must be edited manually if array length

//changes)

void print\_list(int \* arr, int size); //prints all the numbers in the list

void ascend\_bubble\_sort(int \* arr, int size); //sorts a list in ascending order with

//the bubble sort algorithm

void swap\_num(int& a, int& b); //swap function (for bubble sort)

int main()

{

const int SIZE = 6;

int list[SIZE];

cout << "BEFORE: " << endl;

cout << "--------" << endl;

init\_num\_list(list);

print\_list(list, SIZE);

ascend\_bubble\_sort(list, SIZE);

cout << "AFTER: " << endl;

cout << "-------" << endl;

print\_list(list, SIZE);

system("pause");

return 0;

}

//This function assumes the list length is six and must be

//changed manually if list length changes

void init\_num\_list(int \* arr)

{

arr[0] = 9;

arr[1] = 4;

arr[2] = 36;

arr[3] = 11;

arr[4] = 21;

arr[5] = 17;

}

//This function prints all integers in an int list

void print\_list(int \* arr, int size)

{

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

{

cout << arr[i] << endl;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CONTINUES ON NEXT PAGE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//This function uses the bubble sort algorithm to sort an

//array in ascending order

void ascend\_bubble\_sort(int \* arr, int size)

{

bool do\_swap = true;

//For each position in the array and while we need to swap values

for(int i = 0; i < size && do\_swap; ++i)

{

//Reset swap flag each iteration of the outer loop

do\_swap = false;

//Go through each value in the list

for(int j = 0; j < size-1; ++j)

{

//If next position is less than the current, swap them

if(arr[j+1] < arr[j])

{

swap\_num(arr[j+1], arr[j]);

do\_swap = true; //swap occurred

}

}

}

}

//swap function

void swap\_num(int& a, int& b)

{

int temp = a;

a = b;

b = temp;

}

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

2. Use pictures to show each step for sorting the following list in **ascending order** by bubble sort.

|  |  |  |  |
| --- | --- | --- | --- |
| 17 | 4 | 9 | 3 |

3. Use pictures to show each step for sorting the following list in **descending order** by selection sort.

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 7 | 9 | 13 |

4. Use pictures to show each step for finding the value "27" using the sequential search algorithm.

|  |  |  |  |
| --- | --- | --- | --- |
| 17 | 4 | 9 | 3 |

5. Use pictures to show each step for finding the value "27" using the binary search algorithm.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5 | 18 | 43 | -4 | 11 | 33 | 99 |

6. Use pictures to show each step for finding the value "27" using the binary search algorithm.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 6 | 9 | 14 | 15 | 27 | 33 |