|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Part** | **1** | **2** | **3** | **4** | **Total** |
| *maximum* | **25** points | **25** points | **25** points | **25** points | **100**G101010 pointsG |
| ***Your Score*** |  |  |  |  |  |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Searching, Sorting and Complexity Analysis**

Reading Assignment: Thoroughly read Chapter 3 in the course textbook.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part 1 Glossary Terms**

Define, in detail, each of these glossary terms from the realm of computer programming logic and design and computer topics, in general. If applicable, use examples to support your definitions. Consult your notes or course textbook(s) as references or by visiting Web sites such as: [**http://www.ask.com**](http://www.ask.com),[**http://www.bing.com**](http://www.bing.com), [**http://www.webopedia.com**](http://www.webopedia.com)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(a) Binary Search Routine**

|  |
| --- |
| A binary search is a ‘divide and conquer’ search algorithm that finds a midpoint (or, “pivot”) between a list’s start position (index 0) and end position (len(list) -1), and then depending on whether the item at the midpoint index is greater or less than the item being searched for, will move the midpoint to the left or to the right (adding or subtracting the pivot index by 1), until it finds the item it is looking for (or fails).   A smarter binary search uses a similar mechanism, but instead of moving the pivot index left or right by 1, it finds the midpoint between the current pivot and the leftmost boundary (if the search term is less than the item in the pivot index) or rightmost boundary (if the search term is greater than the item in the pivot index) and sets that point as the next pivot. It then repeats this process until it finds the search item and stops (or fails to find the item after exhausting each item in the list).  Binary search assumes that the list is already sorted. It’s worst case complexity is O(log­­­­­2 N). |

**(b) Bubble Sort**

|  |
| --- |
| Bubble sort is a simple but expensive sorting algorithm that starts at the beginning of a list, steps through the list and compares the item at the index *i* with the item at the index *i + 1,* and if they are not in the right sort order, it swaps them. This “bubbles” the largest items (or smallest, if you are doing a descending sort) to the end of the list. It then “repeats the process from the beginning of the list and goes to the next-to-last item, and so on, until it begins with the last item. At that point, the list is sorted” (Lambert 66).  Bubble sort has a quadratic level of complexity O(n^2), since it loops through a list len(list)^2 times. |

**(c) Complexity Analysis**

|  |
| --- |
| Complexity analysis is the art of determining how expensive, in terms of number of times instructions are executed, an algorithm could be in the worst, average, and best case scenario (though algorithms are typically ascribed their Big O order based on their worst case scenario).   Complexity analysis results in a Big O notation “order of complexity”, whose calculation runs from e.g. constant O(k), logarithmic O(log n), linear O(n), quadratic O(n^2), polynomial O(n^6), and exponential O(2^n).  When performing complexity analysis, it is important to remember that in some situations, an expensive (say, O(n^2)) algorithm doesn’t necessarily mean it is better or worse to the end user, if the data set is small enough.   Also, when performing complexity analysis, one gives only the dominant term in the order of magnitude, so e.g., if I had a complexity of 3n^2 + 6, I would only ascribe the quadratic order of magnitude O(n^2). |

**(d) Insertion Sort**

|  |
| --- |
| Insertion sort is a O(n^2) sorting algorithm that steps through a list and compares the current item at index *i* in the outer loop with each item at index *j*  (which is *i – 1)* in the inner loop, and inserts items into the correct position. |

**(e) Selection Sort**

|  |
| --- |
| Selection sort starts at the first position in a list, searches the list for the position of the smallest item, and if that item is not in the first position, it swaps the items at those positions… it then returns to the second position and repeats this process until it reaches the last position (Lambert 64).   Selection sort is O(n^2) in all scenarios. |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part 2 True / False Exercises**

For each of these exercises, enter True or False in the spaces provided.

**FALSE**  **(1)** Processes with linear behavior perform more work than those with quadratic behavior.

**TRUE** **(2)** The amount of work of a logarithmic algorithm is proportional to the log 2 of the problem size.

**TRUE** **(3)** The sequential search algorithm performs more work to find a target at the beginning of a list than at the end of the list.

**TRUE** **(4)** In a bubble sort, each pass through the main loop selects a single item to be moved.

**FALSE. First two numbers are 0 and 1** **(5)** The first two numbers in the Fibonacci sequence are 1s , and each number after that is the sum of the previous two numbers.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part 3 Multiple Choice Exercises**

Select the correct response or responses.

**(1)** What type of algorithm is list indexing an example of?

(a) constant - time (b) logarithmic – time

(c) exponential - time **(d) linear - time**

**(2)** In the following code, what is the algorithm's complexity?

**minIndex = 0**

**currentIndex = 1**

**while currentIndex < len(lyst) :**

**if (lyst[currentIndex] < lyst[minIndex]) :**

**minIndex = currentIndex**

**currentIndex += 1**

**return minIndex**

(a) *O* ( *n* 2 ) **(b) *O* ( *n* )**

(c) *O* ( log 2 ( *n* ) ) (d) *O* ( 2 )

**(3)** What is the worst - case complexity of a binary search?

(a) *O* ( *n* 2 ) (b) *O* ( *n* )

(c) ***O* ( log 2 ( *n* ) )** (d) *O* ( 2 )

**(4)** What term best describes the following code?

**x = myList[i]**

**myList[i] = myList[j]**

**myList[j] = x**

(a) sequential search (b) binary sort

(c) selection algorithm **(d) swap function**

**(5)** What type of algorithm is the following code?

**n = len(myList)**

**while n > 1 :**

**i = 1**

**while i < n :**

**if (myList[i] < myList[i - 1]) :**

**swap(myList, i, i - 1)**

**i += 1**

**n -= 1**

(a) linear sort **(b) bubble sort**

(c) insertion sort (d) selection sort

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part 4 Programming Exercises**

**(1)** **( Selection Sort )**

def selectionSort(*lyst*):

i = 0

*# do n - 1 searches*

*while* i < len(*lyst*)-1:

*# for the smallest*

minIndex = i

j = i + 1

*while* j < len(*lyst*):

*if* *lyst*[j] < *lyst*[minIndex]:

minIndex = j

j += 1

*if* minIndex != i:

swap(*lyst*, minIndex, i)

print(*lyst*)

i += 1

lyst = [4, 8, 19, 2, 28, 21]

selectionSort(lyst)

**OUTPUT**  
[2, 8, 19, 4, 28, 21]

[2, 4, 19, 8, 28, 21]

[2, 4, 8, 19, 28, 21]

[2, 4, 8, 19, 28, 21]

[2, 4, 8, 19, 21, 28]

**(2)** **( Linear Search )**

A linear search starts searching with the first item in the list and compares it to the search criteria. This process continues until an item is met or until the set ends.

The pseudocode for the Linear Search is given here.

**[ Linear Search ]**

**1 ask for the item that is to be searched**

**2 for each item in the list check**

**3 if item\_in\_list = item being searched for then**

**4 report position in list**

**5 end if**

**6 end for loop**

For the list below, how many comparisons would be made when the value to be searched is 10 ?

{ 2 , 3 , 4 , 10 , 40 , 72 }

**Answer: Four comparisons would be made**

**(3)** **( Binary Search )**

*from* tabulate *import* tabulate

def binarySearch(*target*, *sortedLyst*):

'''note! This assumes that list is already sorted!!! '''

iterationTable = []

left = 0

right = len(*sortedLyst*) - 1

*while* left <= right:

midpoint = (left + right) // 2

iterationTable.append([left, right, midpoint])

*if* *target* == *sortedLyst*[midpoint]:

print(tabulate(iterationTable, *headers*=[

"Left", "Right", "Midpoint"], *tablefmt*='github', *numalign*="left"))

*return* f"Index of {*target*} is {midpoint}"

*elif* *target* < *sortedLyst*[midpoint]:

right = midpoint - 1

*else*:

left = midpoint + 1

print(tabulate(iterationTable, *headers*=[

"Left", "Right", "Midpoint"], *tablefmt*='github', *numalign*="left"))

*return* -1

testLyst = [12, 18, 23, 25, 29, 32, 35, 40, 58, 66]

binarySearch(18, testLyst)

**OUTPUT**| Left | Right | Midpoint |

|--------|---------|------------|

| 0 | 9 | 4 |

| 0 | 3 | 1 |

'Index of 18 is 1'

**(4) ( Time Complexity )**

Complete the following Time Complexity table, where *n* is the number of items to be processed. Which column or columns has the smallest value at the bottom row? **log­2(n) has the smallest value at the bottom row, and is better than linear time n.**

|  |  |  |  |
| --- | --- | --- | --- |
| *n* | *n* 2 | log 2 ( *n* ) | *n* log 2 ( *n* ) |
|  |  |  |  |
| 10 | 100 | 3.32 | 33 |
| 100 | 10000 | 6.64 | 664 |
| 200 | 40000 | 7.64 | 1528 |
| 500 | 250000 | 8.97 | 4482 |
| 1000 | 1000000 | 9.97 | 9965 |

**(5) ( Big - O Notation )**

A process is modeled by this function: *T* ( *n* ) = 3 *n* 2 + 20

When considering the end - time behavior of this function, what is *O* [*T* ( *n* ) ] , that is the order of magnitude of the process?

**The order of magnitude is O(n^2), because we only take the dominant term in Big O notation.**