Computing

Lesson 6: Comparing Searching Algorithms

Kashif Ahmed

Materials from the Teach Computing Curriculum created by the National Centre for Computing Education

Task 1 - Linear and Binary Searches

Searching for a planet

Izaz has created a program that stores the planets in our solar system. A sample of data is shown in **Figure 1**.

Element	Earth	Jupiter	Mars	Mercury	Neptune	Saturn	Uranus	Venus
Index	0	1	2	3	4	5	6	7

Figure 1

Task 1 - Linear and Binary Searches

State the total number of elements shown in **Figure 1**.

List the planets that will be compared to the planet ‘Neptune’ when Izaz performs a linear search on the data shown in **Figure 1**.

List the planets that will be compared to the planet ‘Neptune’ when Izaz performs a binary search on the data shown in **Figure 1**.

Task 1 - Linear and Binary Searches

State the planet and number of comparisons that would incur the worst-case scenario (highest number of comparisons) for **linear search** on the data shown in **Figure 1**.

State the planet and number of comparisons that would incur the worst-case scenario (highest number of comparisons) for **binary search** on the data shown in **Figure 1**.

Task 1 - Linear and Binary Searches

Explain which search algorithm is most appropriate for finding a planet in **Figure 1**.

Task 2 - Code for Linear Search

An inefficient linear search algorithm

An implementation of a linear search in Python is shown in Figure 1. Read through the code to familiarise yourself with it; don’t worry if you don’t understand all of it yet.

Task 2 - Code for Linear Search

```
def linear_search(items, search_item):
    #  Initialise the variables
    index = -1
    current = 0
    #  Repeat while the end of the list has not been reached
    while current < len(items):
        #  Compare the current item to the item you are searching for
        if items[current] == search_item:
            index = current
        #  Proceed to the next item in the list
        current = current + 1
    return index
```

Figure 1

Task 2 - Code for Linear Search

State the line number where iteration is first used in **Figure 1**.

Identify one list that is used in **Figure 1.**

Describe what happens when line 11 is omitted from the algorithm in **Figure 1.**

Task 2 - Code for Linear Search

Explain why index needs to be initialised in **Figure 1.**

Explain why the algorithm in **Figure 1** is a function and not a procedure.

Task 2 - Code for Linear Search

Complete the trace table below using the algorithm in **Figure 1** when **items** is the list are

['Reg', 'Chloe', 'Steph', 'Ahmed', 'Keira', 'Neelu'] and the search_item is 'Keira'.

The first two passes have been completed for you.

Task 2 - Code for Linear Search

Line	index	current	items[current]	**Condition**
2	-1			
3		0		
4				True
5			Reg	False
7		1		
4				True
5			Chloe	False
7		2		

Task 2 - Code for Linear Search

Line	`index`	`current`	`items[current]`	**Condition**

Task 2 - Code for Linear Search

Line	`index`	`current`	`items[current]`	**Condition**

Task 2 - Code for Linear Search

A more efficient linear search algorithm

Figure 2 is a more efficient version of a linear search than the one shown in **Figure 1**.

```
def linear_search(items, search_item):
    # Initialise the variables
    index = -1
    current = 0
    found = False
    # Repeat while the end of the list has not been reached
    # and the search item has not been found
    while current < len(items) and found == False:
        # Compare the current item to the item you are searching for
        if items[current] == search_item:
            index = current
            found = True
        # Proceed to the next item in the list
        current = current + 1
    return index
```

Figure 1

Task 2 - Code for Linear Search

State the data type of the variable **found** in **Figure 2**.

The identifier found is a better choice for this variable than **f**. **Give** one reason why.

State one advantage of the algorithm in **Figure 2** compared to that in **Figure 1**.

Task 2 - Code for Linear Search

Describe what it means if the function in **Figure 2** returns a value of -1.

State three advantages of implementing the algorithm in **Figure 2** as a subroutine.

Task 3 - Code for Binary Search

A binary search algorithm

An implementation of a binary search in Python is shown in Figure 1. Read through the code to familiarise yourself with it; don't worry if you don't understand all of it yet.

```python
def binary_search(items, search_item):
    index = -1          #  Initialise the variables
    first = 0
    last = len(items) - 1
    found = False
    while first <= last and found == False:#  Repeat while there are still items  item has not been found
        midpoint = (first + last) // 2   #  Find the middle item (midpoint) between first and last
        if items[midpoint] == search_item:  #  Compare the item at the midpoint to the search item
            index = midpoint
            found = True
        elif items[midpoint] < search_item:
            first = midpoint + 1  #  Focus on right half of range
        else:
            last = midpoint - 1   #  Focus on the left half of range
    return index
```

Figure 1

Task 3 - Code for Binary Search

State the data type of the variable **found** in **Figure 1**.

Which of the following statements is **true**?

a) The algorithm in **Figure 1** uses nested iteration.

b) The algorithm in **Figure 1** uses indefinite iteration.

c) The algorithm in **Figure 1** will loop infinitely.

d) The algorithm in **Figure 1** uses nested selection.

Task 3 - Code for Binary Search

Explain why the calculation of midpoint in line 9 uses floor division.

Task 3 - Code for Binary Search

Complete the trace table below using the algorithm in **Figure 1** when **items** in the list are

['Ahmed', 'Chloe', 'Keira', 'Olivia', 'Neelu', 'Reg', 'Steph', 'Zak'] and the **search_item** is 'Olivia'.

The first pass has been completed for you.

Task 3 - Code for Binary Search

Line	index	first	last	found	midpoint	items[midpoint]	**Condition**
2	-1						
3		0					
4			7				
5				False			
6							True
7					3		
8						Olivia	False
11						Olivia	True

Task 3 - Code for Binary Search

Line	`index`	`first`	`last`	`found`	`midpoint`	`items[midpoint]`	**Condition**
12		4					

Task 3 - Code for Binary Search

Line	index	first	last	found	midpoint	items[midpoint]	**Condition**