

Algorithms and Data Structures

Author: Tomos Leggett · **Student Code:** 1901852 · **Module Code:** ACCA5032

Algorithms and Data Structures

Part A

Question 1

- The benefits of arrays

- The drawbacks of arrays

- The benefits of Linked Lists

- The drawbacks of Linked Lists

Question 2

- The preferable data structure

- Evaluation of performance

Question 3

- Demonstration of the construction of a binary search tree

- The need to maintain a balanced binary tree

Part B

Question 1

Question 2

Part C

- How the Minimax Algorithm works

References

Part A

Question 1

The benefits of arrays

A good benefit of using arrays is its ability to represent multiple data items, of the same type, by a single name. For example, an array of letters in the alphabet can be represented by a single array, called 'alphabet' with the char datatype. [1]

Another good benefit of using arrays is that they are extremely easy to debug. Unlike linked lists, where validating that an index is valid is extremely time consuming and complex comparatively, arrays use an index position value, which can be traversed easily compared to other methods of storing lists of elements. [2]

The index of an array is simply a variable, which means that arrays can be accessed in many different ways. This means that arrays can be accessed randomly, which provides greater functionality, especially in video game development. [3]

As arrays allocate memory when they are declared, they aren't susceptible to a memory overflow or memory leak issue. An array has a set index, and cannot store data beyond that index, which means there is a limit to how much data can be stored in that array, and as the memory is allocated when the array is declared, the developer has a more liberal control over memory management. [4]

Arrays are also fundamental to other methods of storing a list of elements such as alternate data structures like linked lists, stacks, queues, trees or even graphs. Arrays are fundamental to these types of data structures. [5]

The drawbacks of arrays

One major drawback of using arrays to store data is the requirement to set the number of elements that need to be stored in the array prior to storing the data itself. This means arrays cannot be used if the number of elements is flexible and only a set amount of elements can be stored in an array once declared in memory, as the memory is already allocated, it is simply impossible to change the size of the array once it has been declared. [6]

Arrays are homogeneous, which is another major disadvantage of storing data in arrays. Homogeneous, by definition, means that arrays can only store one specific type of data. This means that the data type needs to be declared initially and all elements stored in the array must adhere to that particular set data type. If an array is stored as an integer array, then elements cannot contain character(s), objects or other data types that aren't valid integers. [7]

Arrays are also memory allocated, which means that when they are declared in memory, the entire array reserves its place in memory. If a developer were to allocate a larger array with more elements than maybe necessary, the unused data in that array is needlessly wasted. As arrays are static, their size cannot be adjusted to save memory, as it has already been declared, which makes it a terrible option for storing a flexible amount of data. [8]

Arrays are stored in consecutive memory locations once they're declared, which makes the operation of inserting and deleting elements in an array costly when it comes to memory management. [9]

The benefits of Linked Lists

As discussed above, arrays can only contain a limited set of data, based on the size of the array, which is defined prior to when it is declared and allocated in memory. With Linked List, the opposite occurs. Linked Lists can allocate and deallocate memory, which makes them entirely dynamic compared to arrays. This is especially useful for data that has a variable index size, where the developer is unaware of how much memory needs to be allocated to this list of data. [10]

Inserting and deletion operations are also easier with linked lists, as elements don't need to be physically changed around or re-organised unlike with arrays. Since linked lists use pointers to point the computer towards the data, a linked list simply changes the pointer to another element's location. If an element is deleted, the pointer is simply moved to another element. [11]

Linear data structures (such as a stack or a queue) are usually implemented using linked lists. [12]

The drawbacks of Linked Lists

Linked Lists have the opposite effect compared to arrays when it comes to traversing, with arrays, you have an index value, which can be randomly accessed (i.e. Names[5] will traverse automatically to the 5th element in the 'Names' array.) However, Linked Lists must be traversed manually, element by element. Performance is therefore heavily impacted as the processor needs to sort through individual elements one by one, and depending on the position of the element, this can take a long time. [13]

Because Linked Lists are dynamically allocated, it is not possible to traverse them randomly. Unlike arrays, where their index is a variable, and that variable can be accessed randomly, linked lists must be traversed manually, and don't have an 'index' to each element. [14]

Question 2

The preferable data structure

The brief mentions that the data analysts themselves wish to record thousands of **new** names that are inserted every day. The brief doesn't specify that there will be a set number of names, and as arrays need a set amount of elements, an array would not be appropriate in this particular case. A linked list can contain elements of any data type, unique or otherwise (i.e. data can be repeated etc.) and sorted.

Evaluation of performance

One major issue with a linked list is that there are no particular indexes, which means the data needs to be sorted through manually, one step at a time. If we were to index each index of names, where the names themselves are stored in a linked list, but the linked lists themselves are sorted via an array where the index of the array is based on the first letter of the name, this creates a conundrum; This hybrid reduces the amount of data we need to sort through, as there is a certain amount of letters in the alphabet, we can use an array, and store the linked list in the array itself based on the first letter of the alphabet, which makes sorting through the list itself a lot more efficient than if each and every single name was stored into one list. However, some letters are more common than others, for example, names beginning with M, J and S are quite common, and sorting through them might heavily impact performance especially considering there might be thousands of names beginning with these 3

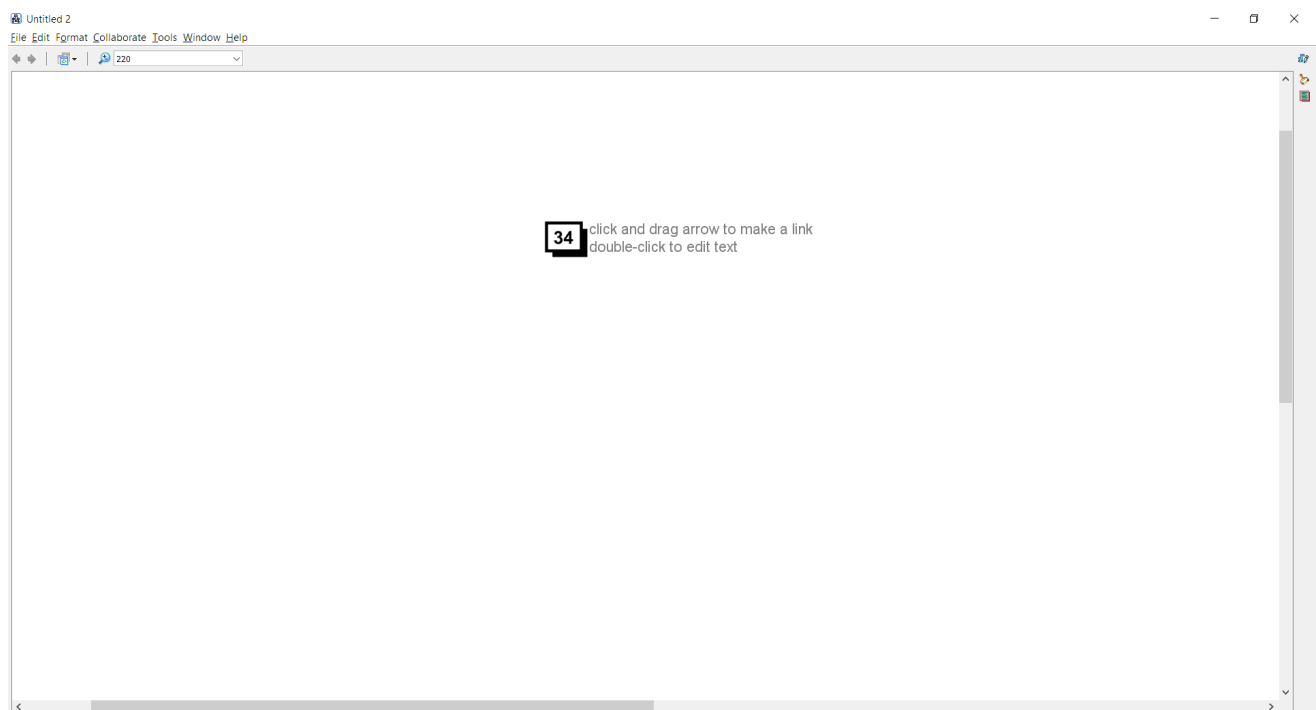
letters, making sorting through them a huge performance impact. Another issue is the way computers sort a list of strings based on alphabetical order; a computer will sort names based on the first letter of their name, but if there are several names beginning with, for example, the letter 'a', the next letter in that particular string (the second letter of that name) will be used to sort instead. If there are a series of names, each with 5 letters very similar, organising these names alphabetically will need more computational power which is another cause for impact on performance.

Question 3

Demonstration of the construction of a binary search tree

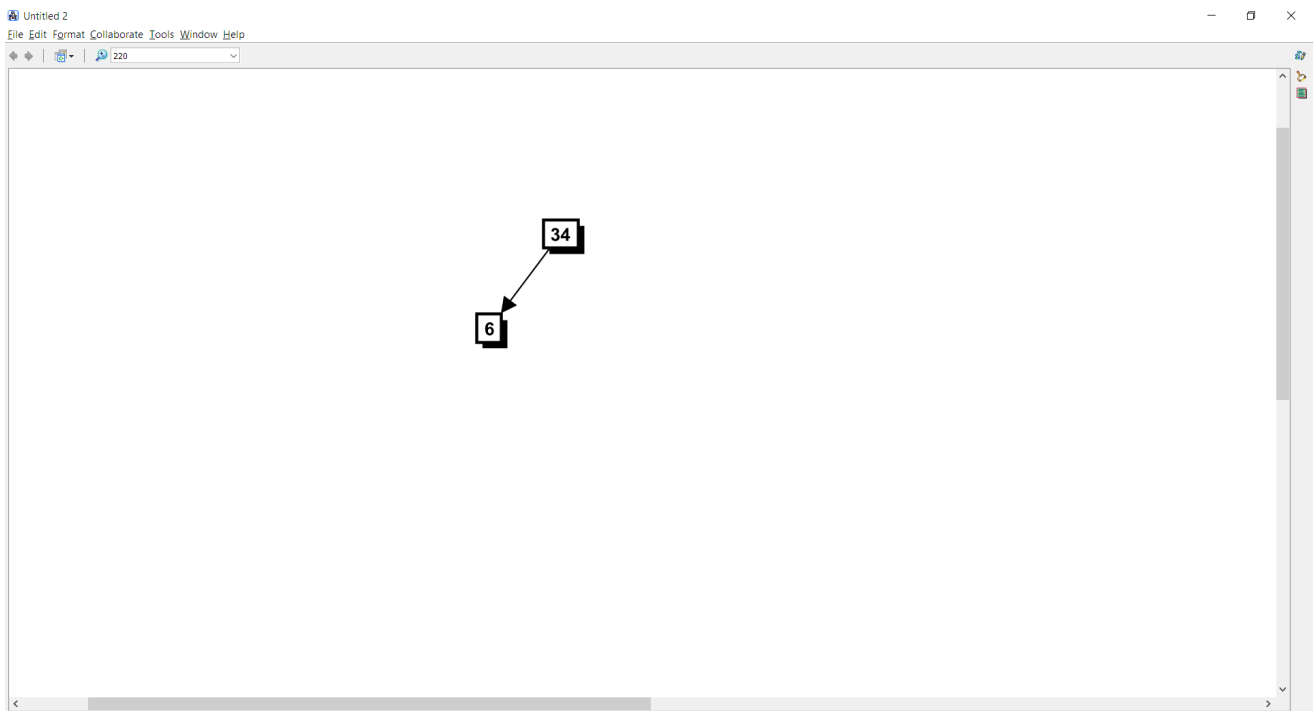
```
1 Values remaining: 34, 6, 7, 18, 45, 12, 3, 26
2 Values used: None
3 Current value: 34
```

To begin the construction of the binary tree, the first element of the array is placed as the root node:



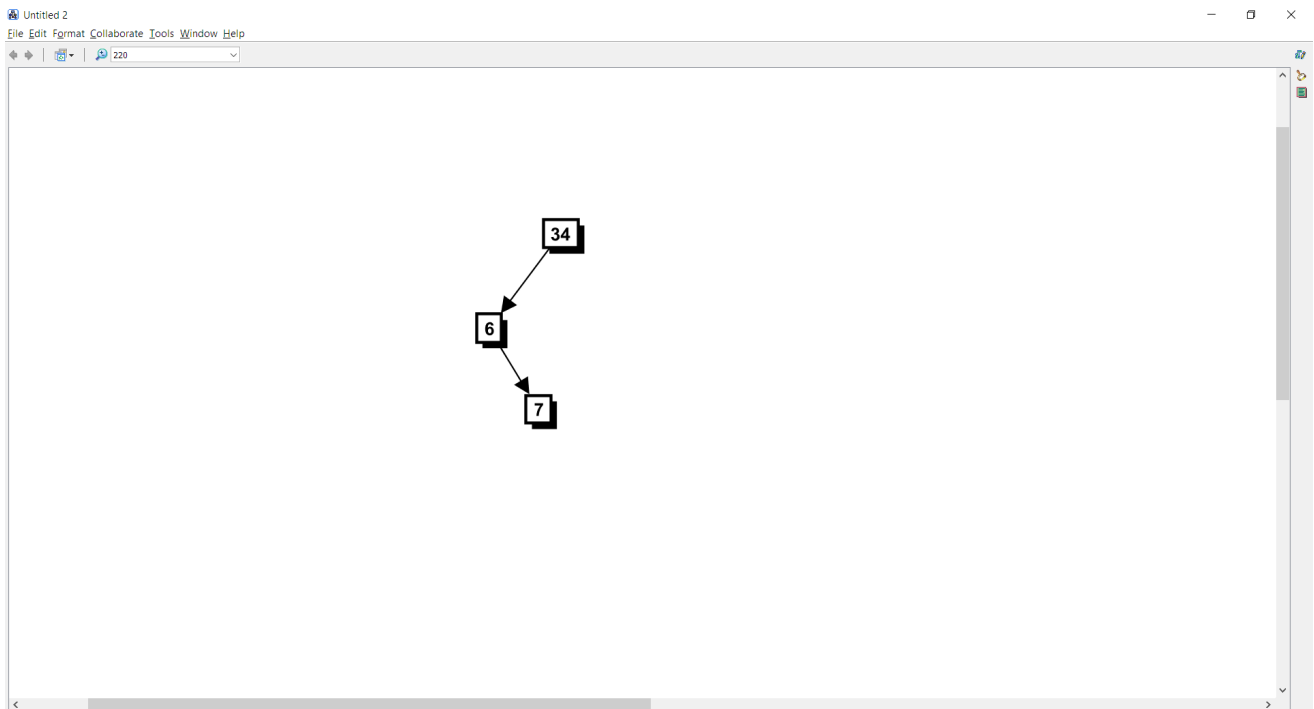
```
1 Values remaining: 6, 7, 18, 45, 12, 3, 26
2 Values used: 34
3 Current value: 6
```

The next value is 6. As 6 is less than 34, the node is placed to the left of the root node, like so:



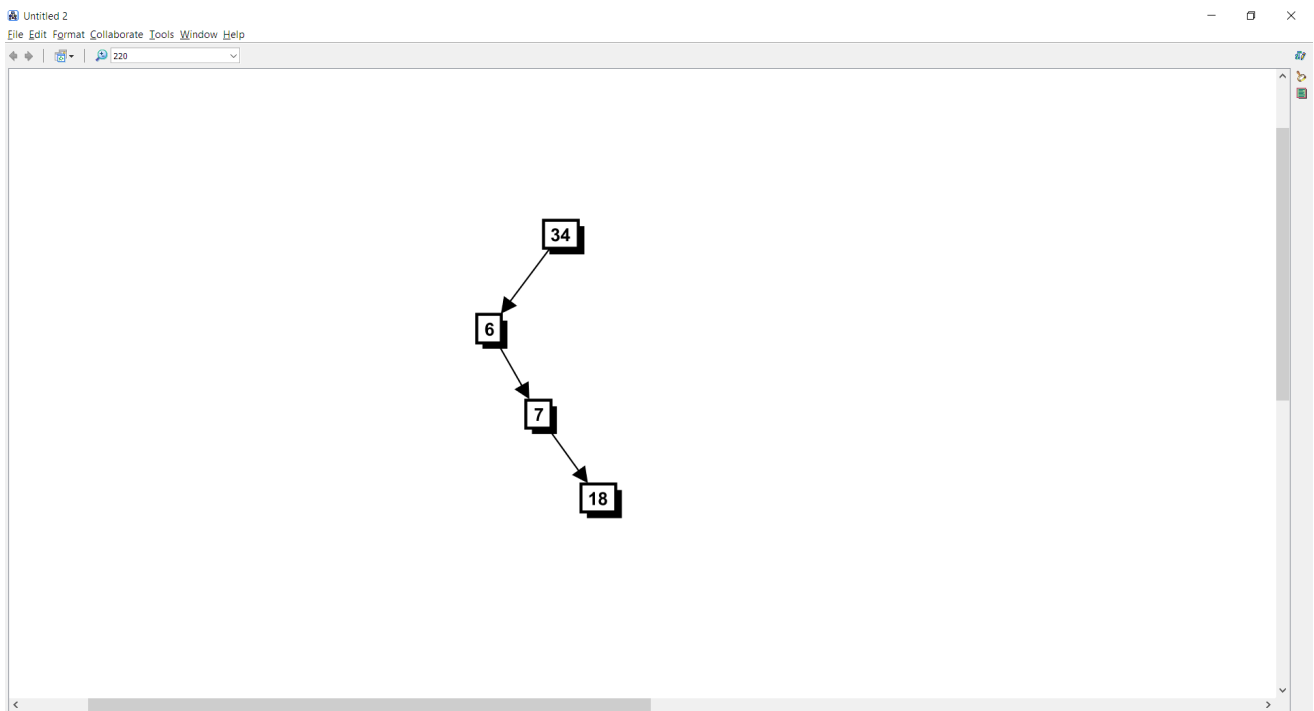
```
1 Values remaining: 7, 18, 45, 12, 3, 26
2 Values used: 34, 6
3 Current value: 7
```

The next value is 7. As 7 is less than 34, it'll be placed on the left branch of the root node. As it is greater than 6, the node will be placed on the right side of the 6 node, like so:



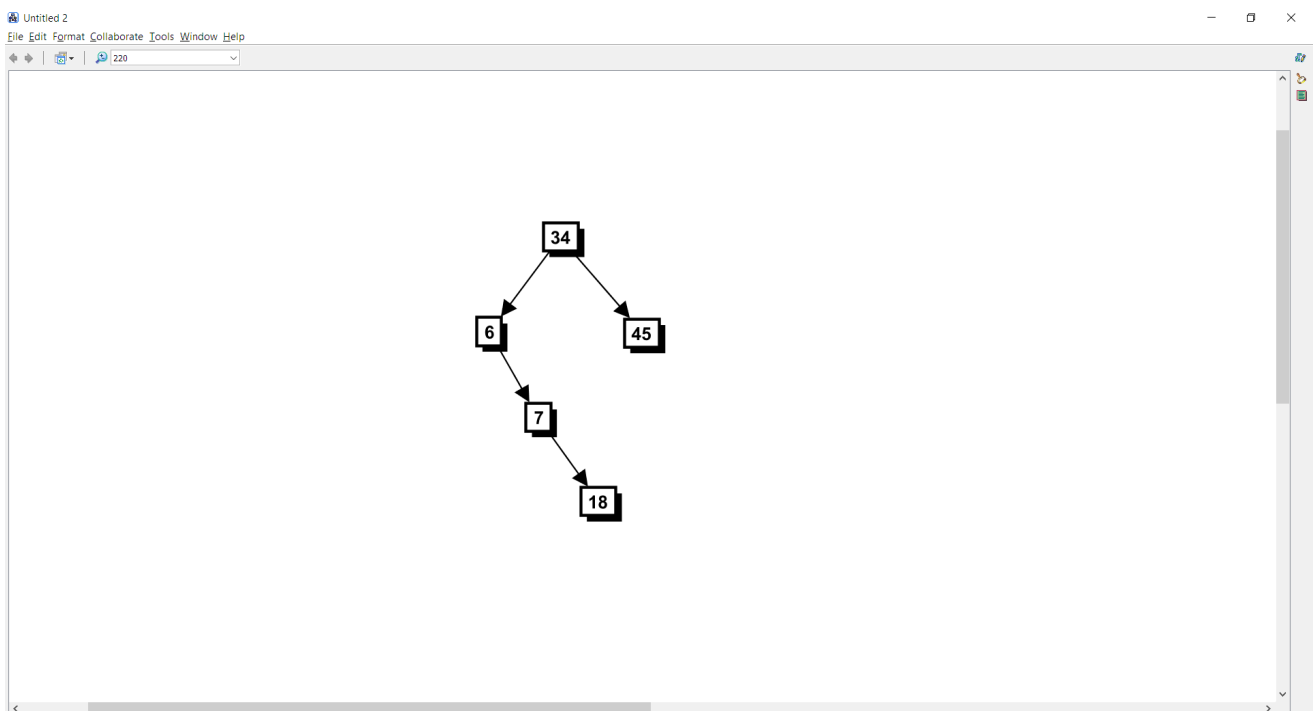
```
1 Values remaining: 18, 45, 12, 3, 26
2 Values used: 34, 6, 7
3 Current value: 18
```

The next value is 18. 18 is less than 34 so it'll be placed on the left of the root node, but it is greater than 6, so it'll be placed on the right of the 6 node. It is also greater than 7, so it will be placed on the right of the 7 node as well, like so:



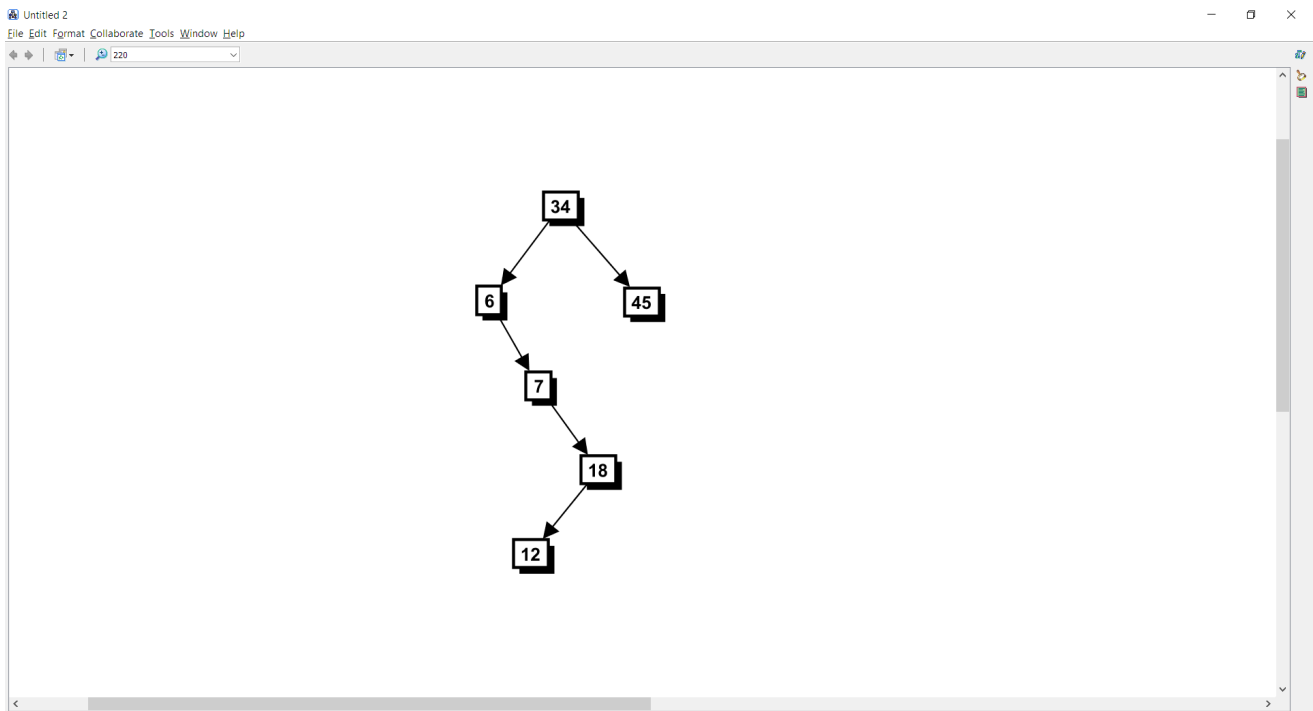
```
1 Values remaining: 45, 12, 3, 26
2 Values used: 34, 6, 7, 18
3 Current value: 45
```

The next value is 45. 45 is larger than 34, and will be the first right branch of the root node. As there are no other branches on the right of the root node, the node will be placed in solidarity, like so:



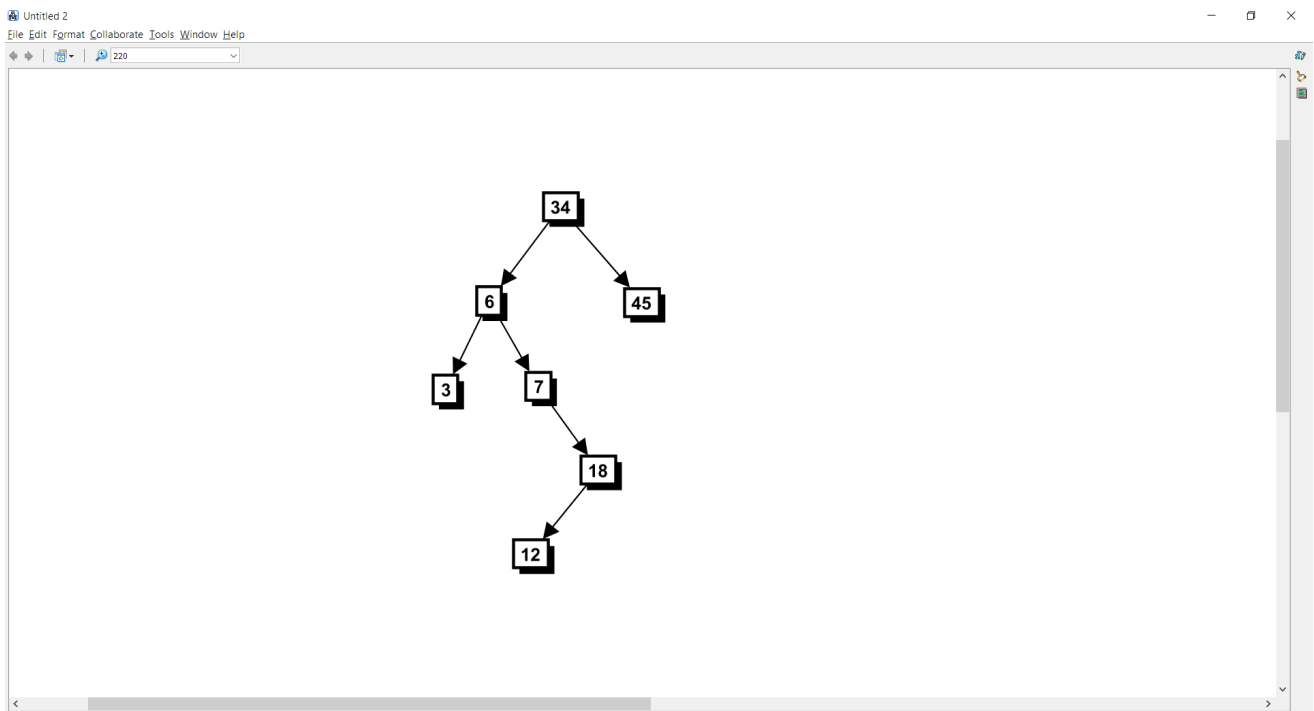
```
1 Values remaining: 12, 3, 26
2 Values used: 34, 6, 7, 18, 45
3 Current value: 12
```

The next value is 12, which is less than 34 and will be placed on the left of the root node. It is however greater than 6, and will traverse until it reaches the 18 node. As it is less than 18, it will be placed on the left of the 18 node, like so:



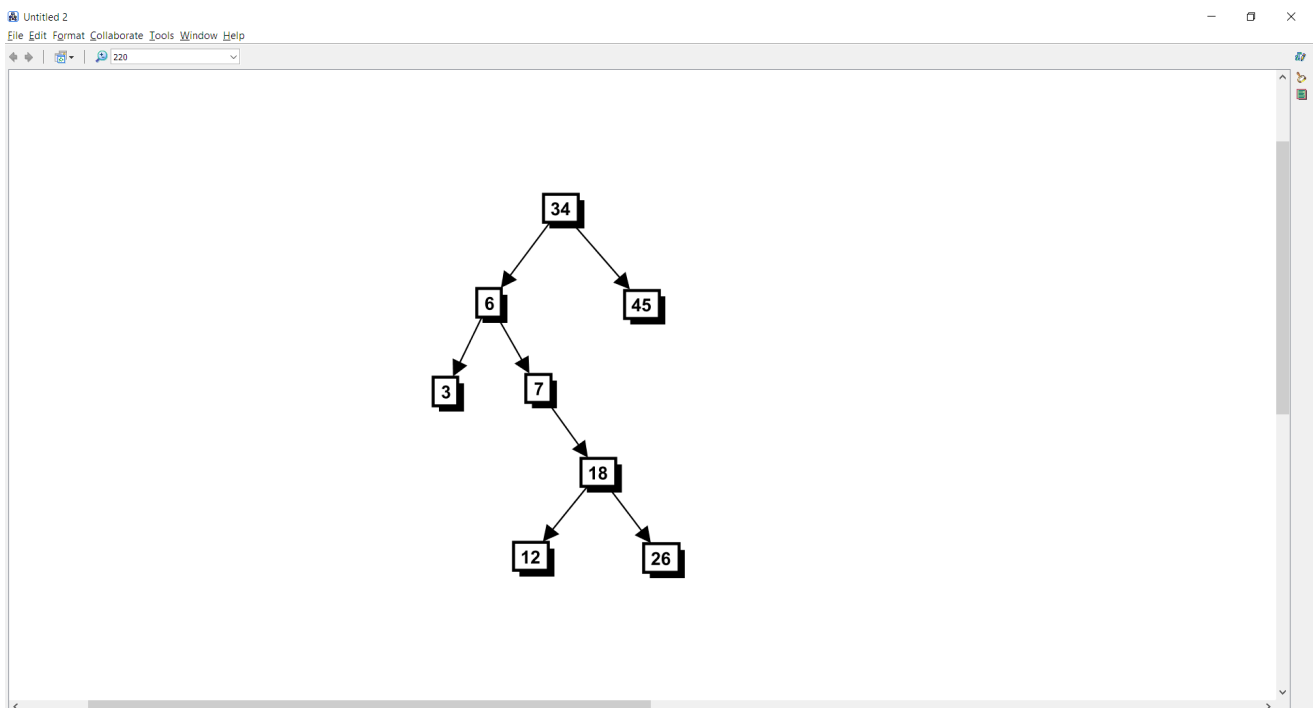
```
1 Values remaining: 3, 26
2 Values used: 34, 6, 7, 18, 45, 12
3 Current value: 3
```

The next value is 3, which is less than 34, so it will be placed on the left of the root node, and will traverse to 6, and will be placed on the left of the 6 node, as it is less than six, like so:



```
1 Values remaining: 26
2 Values used: 34, 6, 7, 18, 45, 12, 3
3 Current value: 26
```

The next value is 26, which is less than 34, and will traverse until it reaches the 18 node, and will be placed on the right side, like so:



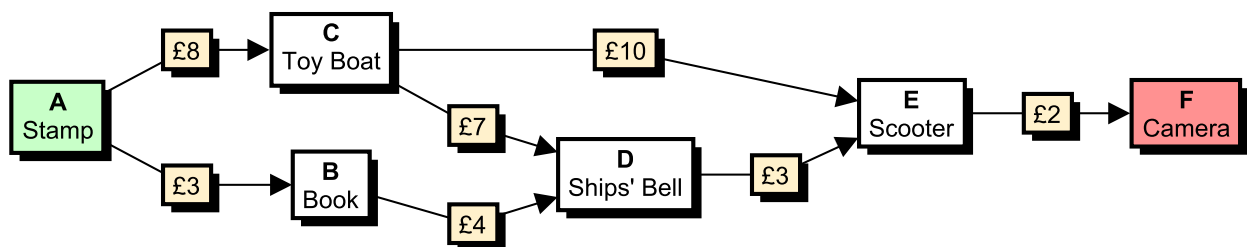
And so, the binary tree is successfully sorted, and can be searched.

The need to maintain a balanced binary tree

It is extremely important to have a balanced binary tree for performance reasons, especially when handling a large array of data. An unbalanced binary tree, which usually has more nodes on a particular side of the tree (for example, the right branch has double the amount of nodes compared to the left node, etc) garners poor performance when sorting. This is because it takes more steps to reach a certain node on an unbalanced binary tree than it does with a balanced binary tree, as there are usually more steps (or hops) to perform. A balanced binary tree however is far more efficient and can increase search performance on a program greatly, and reduce computational power, making the program and the search algorithm a lot more efficient. Algorithms can be used to balance unbalanced binary trees. [15]

Part B

Question 1

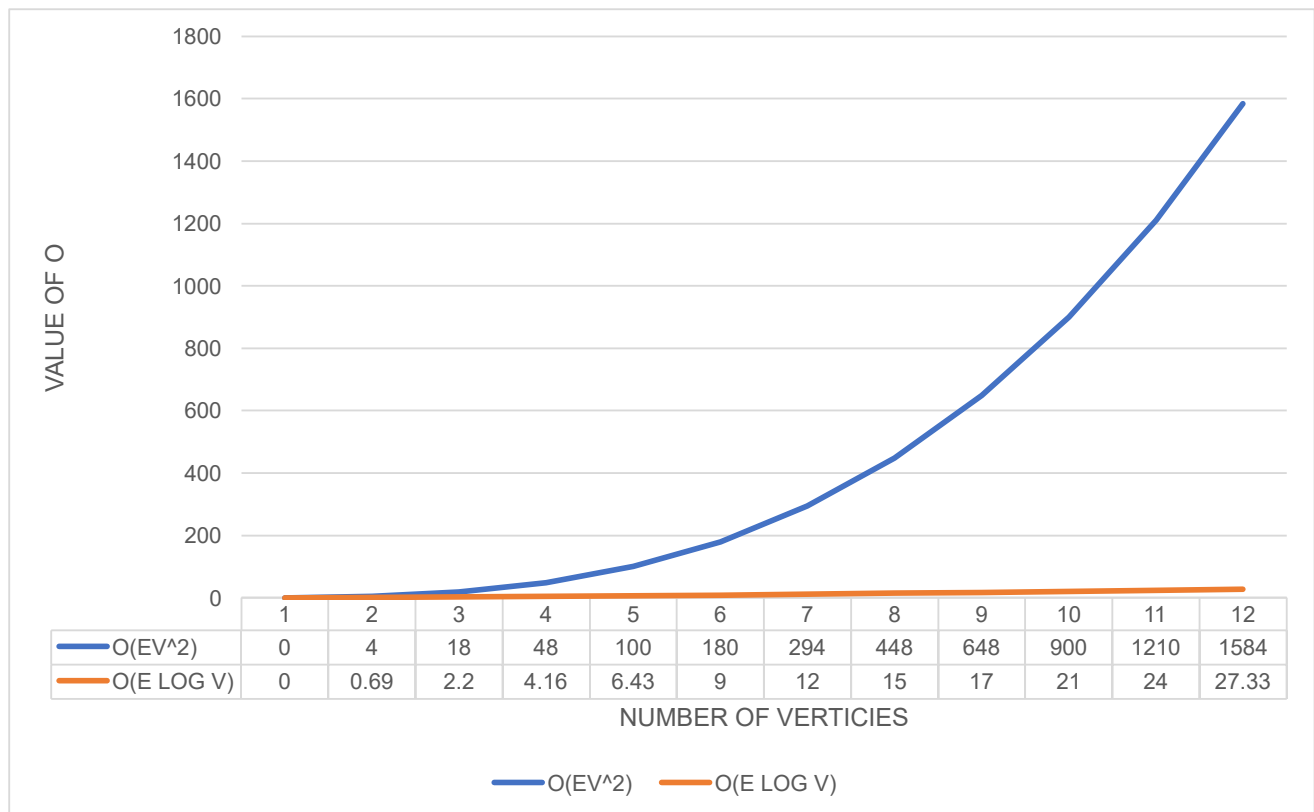


A directed graph was created to illustrate how the lowest cost can be calculated. The yellow represents the edges, and the white represents the vertices. The start and end vertices are colour coded accordingly.

Round	A	B	C	D	E	F
1	0	∞	∞	∞	∞	∞
2		3 (A)	8 (A)	∞	∞	∞
3			8 (A)	7 (B)	∞	∞
4			8 (A)		10 (D)	∞
5					10 (D)	∞
6						12 (E)

A table demonstrating each step. The lowest cost would be to sell the stamp for £3, then the book for £4, then the ships' bell for £3 then the scooter for £2 to finally part exchange for the camera.

Question 2



Above is a graph that showcases the value of O based on the number of vertices increasing from 1 to 12. The blue represents the EV^2 equation, whilst the orange represents the $E \log V$ equation. As you can see, the blue line showcases the value of O increasing exponentially for every added vertices, whilst the orange line barely lifts. This showcases that the priority queue is a far more efficient algorithm.

Part C

How the Minimax Algorithm works

The Minimax Algorithm is an algorithm designed for AI opponents in 2 player games (mostly board games) which aides the decision making processes for the AI opponent by mapping the optimal move before performing it. [16]

The algorithm uses two branches, the 'maximizer' and the 'minimizer'. The 'maximizer' attempts to obtain the highest score whilst the 'minimizer' attempts to gain the lowest score.

The algorithm calculates by attempting all possible moves, storing those results in an array and developing those results into a branch. The left side of the branch is the lowest possible score the player could obtain and the right branch is the highest. The maximizer will always choose the highest and the minimizer will choose the lowest. It then backtracks to the position it is currently in, and makes the decision on what moves are best/worse for the AI to play. [17]

References

- [1] "Minimax Algorithm in Game Theory", *GeeksforGeeks*, 2021. [Online]. Available: <https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/>.
- [2] A. Singh, "Advantages of Array", *EDUCBA*. [Online]. Available: <https://www.educba.com/advantages-of-array/>.
- [3] "Advantages and disadvantages of Arrays", *Smartclass*. [Online]. Available: <http://www.smartclass.co/2011/10/advantages-and-disadvantages-of-arrays.html>.
- [4, 5, 6] "Advantages and Disadvantages of Arrays in C, C++ and Java", *FACE Prep*, 2020. [Online]. Available: <https://www.faceprep.in/data-structures/advantages-and-disadvantages-of-arrays/>.
- [7] "Advantages and Disadvantages of Array in C - GeeksforGeeks", *GeeksforGeeks*, 2020. [Online]. Available: <https://www.geeksforgeeks.org/advantages-and-disadvantages-of-array-in-c/>.
- [8] R. Kakria, "What are Advantages and Disadvantages of arrays?", *Xpode.com*. [Online]. Available: <http://www.xpode.com/ShowArticle.aspx?ArticleId=502>.
- [9] "Advantages and Disadvantages of Array in C Programming", *TutorialCup*. [Online]. Available: <https://www.tutorialcup.com/cprogramming/array-advantages-disadvantages.htm>.
- [10, 11] N. Mishra, "Advantages and Disadvantages of Linked List", *The Crazy Programmer*. [Online]. Available: <https://www.thecrazyprogrammer.com/2016/11/advantages-disadvantages-linked-list.html>.
- [12, 13] "Advantages and Disadvantages of Linked List", *GeeksforGeeks*, 2020. [Online]. Available: <https://www.geeksforgeeks.org/advantages-and-disadvantages-of-linked-list/>.
- [14] "Dev.Interview", *Developer-interview.com*. [Online]. Available: <https://developer-interview.com/p/algorithms/what-is-a-linked-list-and-how-it-works-what-are-advantages-and-disadvantages-of-linked-lists-39>.
- [15] J. Hudgens, "Why do Binary Search Trees Have to be Balanced?", *CronDose*, 2016. [Online]. Available: <https://www.crondose.com/2016/08/binary-search-trees-balanced/>.
- [16] "Mini-Max Algorithm", *www.javatpoint.com*. [Online]. Available: <https://www.javatpoint.com/mini-max-algorithm-in-ai>.
- [17] "Minimax Algorithm in Game Theory", *GeeksforGeeks*. [Online]. Available: <https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/>.