Data Structures and Algorithms  **Binary Search Tree Assignment 4**

1. What is the divide and conquer strategy?

* In divide and conquer approach, the problem is divided into smaller sub-problems and then each problem is solved independently. When we keep on dividing the sub problems into even smaller sub-problems, we may eventually reach a stage where no more division is possible.
* Those "atomic" smallest possible sub-problem (fractions) are solved.
* The solution of all sub-problems is finally merged in order to obtain the solution of an original problem.
* **Divide/Break** -Breaking the problem into smaller sub-problems. Sub-problems should represent a part of the original problem. This step generally takes a recursive approach to divide the problem until no sub-problem is further divisible.
* **Conquer/Solve**

This step receives a lot of smaller sub-problems to be solved. Generally, at this level, the problems are considered 'solved' on their own.

* **Merge/Combine**

When the smaller sub-problems are solved, this stage recursively combines them until they formulate a solution of the original problem. This algorithmic approach works recursively and conquer & merge steps works so close that they appear as one.

2. What is binary search and how does it work?

* Binary search is a fast search algorithm with run-time complexity of Ο(log n).
* This search algorithm works on the principle of divide and conquer.
* For this algorithm to work properly, the data collection should be in the sorted form.
* Binary search looks for a particular item by comparing the middle most item of the collection.
* If a match occurs, then the index of item is returned.
* If the middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item. Otherwise, the item is searched for in the sub-array to the right of the middle item. This process continues on the sub-array as well until the size of the subarray reduces to zero.
* For a binary search to work, it is mandatory for the target array to be sorted.
* The following is our sorted array and let us assume that we need to search the location of value 31 using binary search.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 14 | 19 | 26 | 27 | 31 | 33 | 35 | 42 | 44 |

0 1 2 3 4 5 6 7 8 9

* First, we shall determine half of the array by using this formula −
* mid = low + (high - low) / 2
* Here it is, 0 + (9 - 0 ) / 2 = 4 (integer value of 4.5). So, 4 is the mid of the array.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **10** | **14** | **19** | **26** | **27** | **31** | **33** | **35** | **42** | **44** |

**0 1 2 3 4 5 6 7 8 9**

* Now we compare the value stored at location 4, with the value being searched, i.e. 31. We find that the value at location 4 is 27, which is not a match. As the value is greater than 27 and we have a sorted array, so we also know that the target value must be in the upper portion of the array.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ~~10~~ | ~~14~~ | ~~19~~ | ~~26~~ | ~~27~~ | **31** | **33** | **35** | **42** | **44** |

0 1 2 3 4 5 6 7 8 9

* We change our low to mid + 1 and find the new mid value again.
* low = mid + 1
* mid = low + (high - low) / 2
* Our new mid is 7 now. We compare the value stored at location 7 with our target value 31.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ~~10~~ | ~~14~~ | ~~19~~ | ~~26~~ | ~~27~~ | **31** | **33** | **35** | **42** | **44** |

0 1 2 3 4 5 6 7 8 9

* The value stored at location 7 is not a match, rather it is more than what we are looking for. So, the value must be in the lower part from this location.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ~~10~~ | ~~14~~ | ~~19~~ | ~~26~~ | ~~27~~ | **31** | **33** | ~~35~~ | ~~42~~ | ~~44~~ |

0 1 2 3 4 5 6 7 8 9

* Hence, we calculate the mid again. This time it is 5.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ~~10~~ | ~~14~~ | ~~19~~ | ~~26~~ | ~~27~~ | **31** | **33** | ~~35~~ | ~~42~~ | ~~44~~ |

0 1 2 3 4 5 6 7 8 9

* We compare the value stored at location 5 with our target value. We find that it is a match.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ~~10~~ | ~~14~~ | ~~19~~ | ~~26~~ | ~~27~~ | **31** | ~~33~~ | ~~35~~ | ~~42~~ | ~~44~~ |

0 1 2 3 4 5 6 7 8 9

* We conclude that the target value 31 is stored at location 5.
* Binary search halves the searchable items and thus reduces the count of comparisons to be made to very less numbers.

3. Explain the distinction between a list and a tuple.

|  |  |
| --- | --- |
| List | Tuple |
| It is mutable | It is immutable |
| The implication of iterations is time-consuming in the list. | Implications of iterations are much faster in tuples. |
| Operations like insertion and deletion are better performed. | Elements can be accessed better. |
| Consumes more memory. | Consumes less memory. |
| Many built-in methods are available. | Does not have many built-in methods. |
| Unexpected errors and changes can easily occur in lists. | Unexpected errors and changes rarely occur in tuples. |

4. Can you explain how Python manages memory?

* Memory allocation can be defined as allocating a block of space in the computer memory to a program.
* Memory Allocation in Python are of 2 types
* **stack memory** and **heap memory**
* The methods/method calls and the references are stored in stack memory and all the values objects are stored in a private heap.

**Work of Stack Memory**

* The allocation happens on contiguous blocks of memory. We call it stack memory allocation because the allocation happens in the function call stack. The size of memory to be allocated is known to the compiler and whenever a function is called, its variables get memory allocated on the stack.
* It is the memory that is only needed inside a particular function or method call. When a function is called, it is added onto the program’s call stack. Any local memory assignments such as variable initializations inside the particular functions are stored temporarily on the function call stack, where it is deleted once the function returns, and the call stack moves on to the next task. This allocation onto a contiguous block of memory is handled by the compiler using predefined routines, and developers do not need to worry about it.

**Work of Heap Memory**

* The memory is allocated during the execution of instructions written by programmers. Note that the name heap has nothing to do with the heap data structure. It is called heap because it is a pile of memory space available to programmers to allocated and de-allocate. The variables are needed outside of method or function calls or are shared within multiple functions globally are stored in Heap memory.

5. What is the difference between pickling and unpickling?

* The pickle module is used for implementing binary protocols for serializing and de-serializing a Python object structure.
* **Pickling**: It is a process where a Python object hierarchy is converted into a byte stream.
* **Unpickling**: It is the inverse of Pickling process where a byte stream is converted into an object hierarchy.

6. What are the different types of search algorithms?

* Searching Algorithms are designed to check for an element or retrieve an element from any data structure where it is stored. Based on the type of search operation, these algorithms are generally classified into two categories:
* **Sequential Search**: In this, the list or array is traversed sequentially and every element is checked. For example: Linear Search.
* **Interval Search**: These algorithms are specifically designed for searching in sorted data-structures. These type of searching algorithms are much more efficient than Linear Search as they repeatedly target the center of the search structure and divide the search space in half. For Example: Binary Search.

**Searching Algorithms :**

* Linear Search
* Binary Search
* Jump Search
* Interpolation Search
* Exponential Search
* Sublist Search (Search a linked list in another list)
* Fibonacci Search
* The Ubiquitous Binary Search
* Recursive program to linearly search an element in a given array
* Recursive function to do substring search