Algorithm and Data Structures
Week 6

Endang Wahyu Pamungkas, Ph.D.

Searching

- Searching for data stored in different data structures is a crucial part in every single application.
- There are many different algorithms available to utilize when searching, and each have different implementations and rely on different data structures to get the job done.
- Being able to choose a specific algorithm for a given task is a key skill for developers and can mean the difference between a fast, reliable and stable application.

Membership Operator

- In Python, the easiest way to search for an object is to use Membership Operators - named that way because they allow us to determine whether a given object is a member in a collection.
- These operators can be used with any iterable data structure in Python, including Strings, Lists, and Tuples.
  - **in** - Returns True if the given element is a part of the structure.
  - **not in** - Returns True if the given element is not a part of the structure.

Membership Operator

```
>>> 'apple' in ['orange', 'apple', 'grape']
True
>>> 't' in 'stackabuse'
True
>>> 'q' in 'stackabuse'
False
>>> 'q' not in 'stackabuse'
True
```

Membership Operator

- In most cases we need the position of the item in the sequence, in addition to determining whether or not it exists.
- Membership operators do not meet this requirement.
- There are many search algorithms that don't depend on built-in operators and can be used to search for values faster and/or more efficiently.

Sequential / Linear Search

- Linear search is one of the simplest searching algorithms, and the easiest to understand. We can think of it as a ramped-up version of our own implementation of Python's in operator.
- The algorithm consists of iterating over an array and returning the index of the first occurrence of an item once it is found.
- Linear search is a good fit for when we need to find the first occurrence of an item in an unsorted collection.

Sequential / Linear Search

```
def LinearSearch(lys, element):
    for i in range (len(lys)):
        if lys[i] == element:
            return i
    return -1
```

- The time complexity of linear search is O(n), meaning that the time taken to execute increases with the number of items in our input list.
- Linear search is not often used in practice, because the same efficiency can be achieved by using inbuilt method.

Binary Search

- Binary search follows a **divide and conquer** methodology. It is faster than linear search but requires that the array be sorted before the algorithm is executed.
- Assuming that we're searching for a value **val** in a **sorted array**, the algorithm compares **val** to the value of the middle element of the array, which we'll call **mid**.
- The binary search algorithm can be written either **recursively** or **iteratively**. Recursion is generally slower in Python because it requires the allocation of new stack frames.

Binary Search

Algorithm of Binary Search :

1. If **mid** is the element we are looking for (best case), we return its index.
2. If not, we identify which side of **mid val** is more likely to be on based on whether **val** is smaller or greater than **mid**, and discard the other side of the array.
3. We then recursively or iteratively follow the same steps, choosing a new value for **mid**, comparing it with **val** and discarding half of the possible matches in each iteration of the algorithm.

https://www.youtube.com/watch?v=MFhxShGxHWc

Binary Search

```
def BinarySearch(lys, val):
    first = 0
    last = len(lys)-1
    index = -1
    while (first <= last) and (index == -1):
        mid = (first+last)//2
        if lys[mid] == val:
            index = mid
        else:
            if val<lys[mid]:
                last = mid -1
            else:
                first = mid +1
    return index
```

Binary Search

- Binary Search can work just like linear search and return the first occurrence of the element in some cases.
- If we perform binary search on the array [1,2,3,4,4,5] for instance, and search for 4, we would get 3 as the result.
- Binary search is quite commonly used in practice because it is efficient and fast when compared to linear search. However, it does have some shortcomings, such as its reliance on the // operator.

Binary Search

- We can only pick one possibility per iteration, and our pool of possible matches gets divided by two in each iteration. This makes the time complexity of binary search O(log n).
- One drawback of binary search is that if there are multiple occurrences of an element in the array, it does not return the index of the first element, but rather the index of the element closest to the middle.

Exponential Search

- Exponential search is another search algorithm that can be implemented quite simply in Python. It is also known by the names galloping search, doubling search and Struzik search.
- Exponential search depends on binary search to perform the final comparison of values. The algorithm works by:
  - Determining the range where the element we're looking for is likely to be.
  - Using binary search for the range to find the exact index of the item.

Exponential Search

```
def ExponentialSearch(lys, val):
    if lys[0] == val:
        return 0
    index = 1
    while index < len(lys) and lys[index] <= val:
        index = index * 2
    return BinarySearch(lys[:min(index, len(lys))], val)
```

1,2,4,8,16,32

Exponential Search

- Exponential search runs in O(log i) time, where i is the index of the item we are searching for. In its worst case, the time complexity is O(log n), when the last item is the item we are searching for (n being the length of the array).
- Exponential search works better than binary search when the element we are searching for is closer to the beginning of the array. In practice, we use exponential search because it is one of the most efficient search algorithms

https://www.youtube.com/watch?v=PaGRX7IlaWU&t=5s

Conclusion

- If you want to search through an unsorted array or to find the first occurrence of a search variable, the best option is linear search.
- If you want to search through a sorted array, there are many options of which the simplest and fastest method is binary search.
- If you know that the element you're searching for is likely to be closer to the start of the array, you can use exponential search.

Exercise

- Diketahui sebuah sorted array sebagai berikut ini :
  arr = [2, 3, 5, 7, 8, 9, 13, 15, 17, 18, 20 ,21, 23, 25, 26, 29, 32, 35, 36]
- Jika anda diminta untuk mencari angka 26, jelaskan step by step untuk menemukan angka tersebut dengan algoritma **binary search**.
- Jika anda diminta untuk mencari angka 25, jelaskan step by step untuk menemukan bagian yang akan dieksekusi dengan algoritma **binary search** dengan menggunakan algoritma **exponential search**.