Notes on Chapter 10 - Some Simple Algorithms and Data Structures

Swarup Tripathy *

May 2022

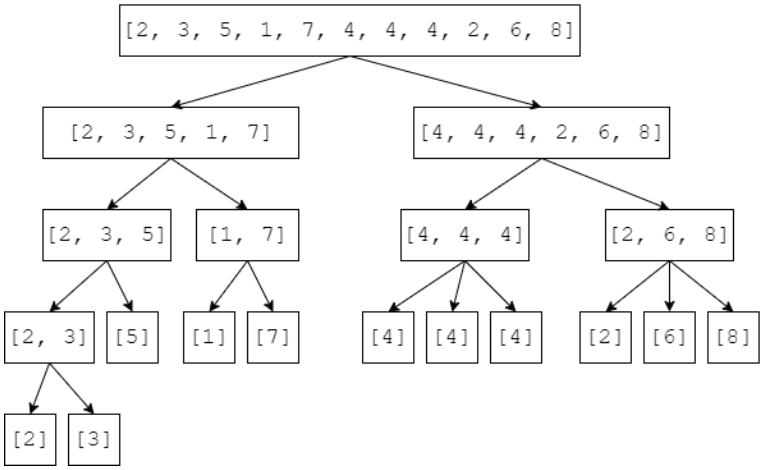
A curated list of important points for my reference.

1. Introduction to Algorithms, by Cormen, Leiserson, Rivest, and Stein, is an excellent source for thous of you not intimidated by a fair amount of mathematics.
2. The key to efficiency is a good algorithm, not clever coding tricks.
3. **SEARCH ALGORITHMS**
   (a) a method for finding an item or group of items with specific properties within a collection of items.
   (b) *BINARY SEARCH*
   - Binary search is similar to the bisection search algorithm
   - Here we rely on the assumption that the list is ordered
   - Pick an index i, that divides the list l roughly in half
   - ask if l[i] == e
   - if not, ask whether l[i] is larger or smaller than e
   - depending upon the answer, search either left or right half of l for e.
4. **SORTING ALGORITHMS**
   - The standard implementation of sorting in most Python implementations runs in roughly O(n*log(n)) time, where n is the length of the list.
   - In most cases, the right thing to do is to use either Python's built in sort method *L.sort()* or its built in function sorted *sorted(L)*
   - *SELECTION SORT*
     – given list [4,2,6,1]
     – a 'while loop' over individual elements

*John V Guttag

- a 'for loop' under 'while loop' traversing over individual elements
- if the 'for loop' element is less than the 'while loop' element, swap it
- Here the complexity of the algorithm will be quadratic in the length of L.

- *MERGE SORT*
  - Also known as Divide and Conquer Algorithm.
  - Breaks down problem into multiple subproblems recursively until they become simple to solve.
  - Solutions are combined to solve original problem
  - O(n*log(n)) is the running time, optimal running time for comparison based algorithms.
  - General Principle
    * Split array in half
    * Call mergeSort on each half to sort them recursively
    * Merge both sorted halves into one sorted array.
    * We continue this until we get arrays of size 1, since arrays of size 1 are always sorted.



    * At the bottom nodes, you can observe arrays of size 1

- The sorting algorithm used in most Python Implementation is called **timsort.**
- Timsort's worst case performance is the same as merge sort's, but on average it performs considerablty better.

5. The relational operators compare the Unicode values of the characters of the strings from the zeroth index till the end of the string. It then returns a boolean value according to the operator used. String Comparison

6. Both list.sort and sorted function can have two additional parameters. The key parameter plays the same role as in our implementation of merge.sort: it supplies the comparison function to be used.

```
L = [[1,2,3],(3,2,1,0),'abc']
print(sorted(L,key=len,reverse=True))
# sorts the element of L in reverse order of length and prints

# Output is
[(3, 2, 1, 0), [1, 2, 3], 'abc']
```

7. **Hash Tables**

- The hash value is an integer which is used to quickly compare dictionary keys while looking at a dictionary.
- Dictionaries use a technique called hashing to do the lookup in time that is nearly indpendent of the size of the dictionary.
- We convert the key to an integer, and then use that integer to index into a list, which can be done in constant time.
- Internal Representation of string 'abc' is

```
>>> list(map(bin,bytearray('abc','utf8')))
    ['0b1100001', '0b1100010', '0b1100011']

>>> ''.join(list(map(bin,bytearray('abc','utf8'))))
    '0b11000010b11000100b1100011'
```

```
'utf8' is the encoding type
bin is Binary conversion
bytearray returns a bytearray object (i.e. array of bytes)
```

- A hash function maps a large space of inputs(e.g. all natural numbers) to a smaller space of outputs(e.g. the natural numbers between 0 and 5000).
- They can be used to convert a large space of keys to a smaller space of integer indices.
- Hash function is a **many-to-one mapping.** i.e., multiple different inputs may be mapped to the same output.
- When two inputs are mapped to the same output, it is called *collision.*
- A good hash function produces a *uniform distribution* i.e., every output in the range is equally probable, which minimizes the probability of collisions.
- The basic idea is to represent an instance of class intDict by a list of *hash buckets* where each bucket is a list of key/value pairs implemented as tuples.