Middle East Technical University | Department of Computer Engineering

CENG 310

Algorithms and Data Structures with Python

Spring 2020-2021

Homework-04 Solution

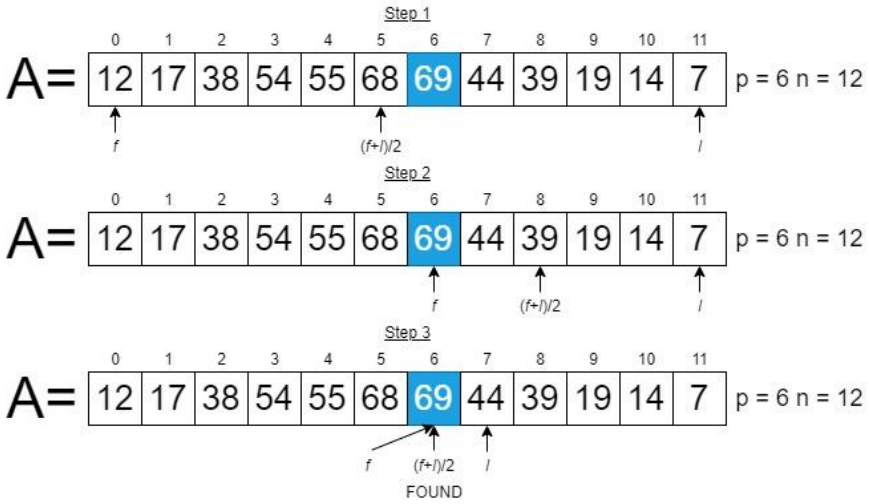
By Mehmet Taha Şahin

Pseudo Code:

```
P_SEARCH(f, l, A) //f = first Index (initially 0), l= last Index(initially n-1), A = array
1  if f == l
2     return f
3  if ((f+l)/2 == f or A[(f+l)/2] > A[(f+l)/2 - 1]) and A[(f+l)/2] > A[(f+l)/2 + 1]
4     return (f+l)/2
5  else if ((f+l)/2 == f or A[(f+l)/2] > A[(f+l)/2 - 1]) and A[(f+l)/2] < A[(f+l)/2 + 1]
6     return P_SEARCH((f+l)/2+1, l, A)
7  else
8     return P_SEARCH(f, (f+l)/2-1, A)
```

The p index at given task can be thought of as a peak point. In other words, while there is an increasing order before this peak point, there is a decreasing order afterwards. This means that both neighbors of the peak point are lower than itself. This can be thought of as binary search logic. At first, it should be checked whether the middle of the array is the peak point. If it is the peak point, that index should be returned. If not, it is necessary to go to the right or left subarray recursively by checking whether it is an ascending or descending order and learning which half of the array the peak is in. If the subarray has become a single element with a shrinkage, it does not need to be controlled anymore, it means that the peak point has already been reached. This is in stopping condition as seen in line 1. Also, integer operation applied for finding middle element. It is "(*f*+*l*)/2" and this operation could be equal to *f* itself. For example, for *f*=6 and *l*=7, middle is also 6. That means middle element is on the left

boundary of subarray and there is no left neighbor for middle element. For that, in line 3 and line 5 this extra condition "(*f*+*l*)/2 == *f* **or**" is added to the algorithm.



As seen in the figure above, when the algorithm is applied, the desired p index is found in 3 steps. In other words, in this example where n is 12, the result was found by looking at 3 separate index locations. This corresponds to 3 units of time.

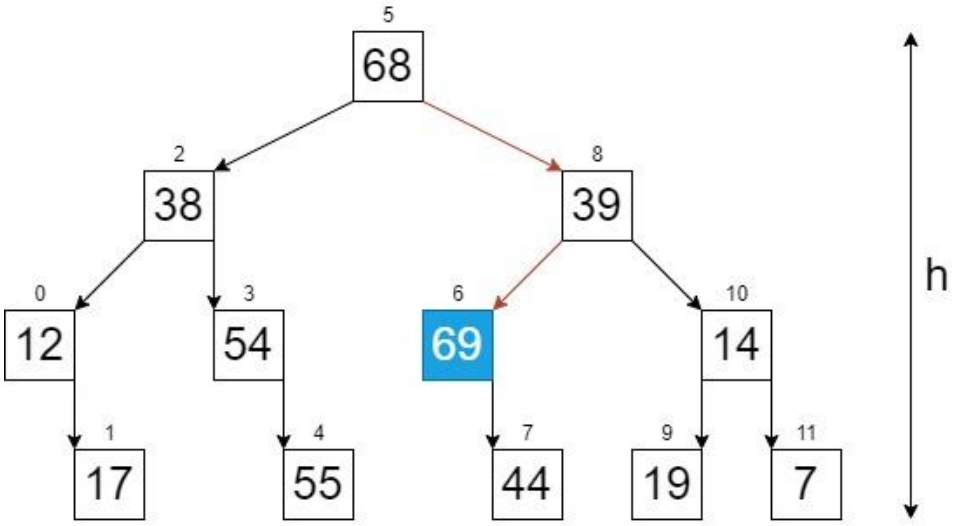


Figure above shows that how binary search like algorithms changes array to tree like structures. The binary search like algorithms sees the array as the tree figure above and chooses one of two ways at each step. Red arrows show the path taken by the algorithm. In worst case, it goes to the bottom of the tree, that is, to its leaves. This corresponds to the height of the tree. Since this is a balanced tree, the height is *logn* in a tree structure with *n* elements. In other words, the complexity of the algorithm is O(*logn*).