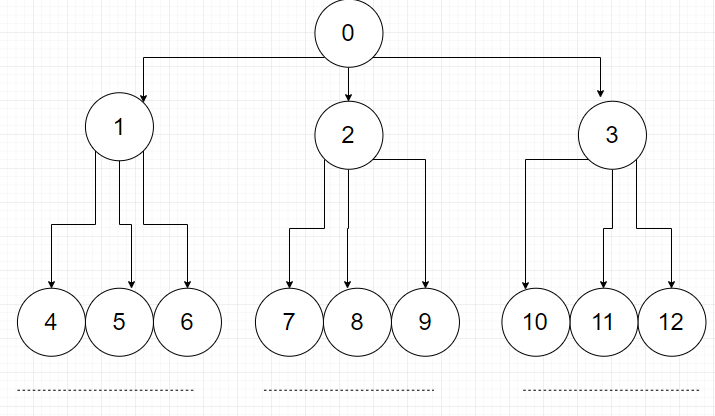
1A

We know if given array A with index i of a node with 2 child, then its parent can be presented as (i/2) , its left child can be presented as 2i, and right child can be presented as 2i+!, so if we have m child for each parent node. Assume the root at A[1] , A[1]’s child listed from A[2] to A[m+1]. The child of A[2] to A[m+1] are listed from A[m+2] to A [m+1+m2]. So for heap node with m child, the parent can be write as (i-1)/2.

1B

Assume we have an array A = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 ……], with each node should have 3 child.(s=3 in this case.)

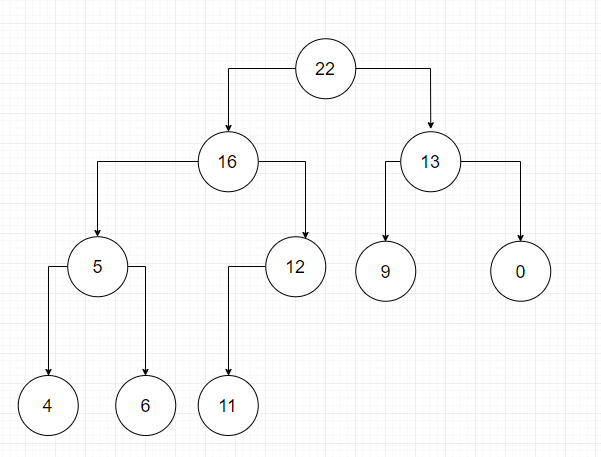
As above shows the node 0 has its 1st child as node 1, its 2nd child as node 2, 3rd child as node 3 .

Node 1 has its 1st child as node 4, its 2nd child as node 5, 3rd child as node 6.

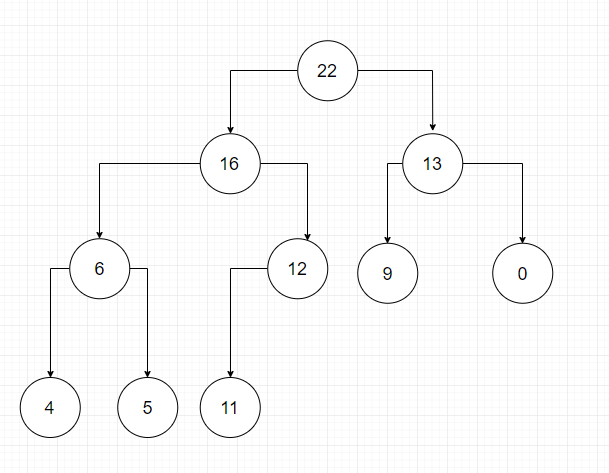
In this case ist node has its 1st child as node si+1, its 2nd child as node si+2, 3rd child as node si+k. (for example node 2 has its 3rd child at 2\*3+3 = 9).

So for kth child of node i, the formula is **si+k** (s-child)

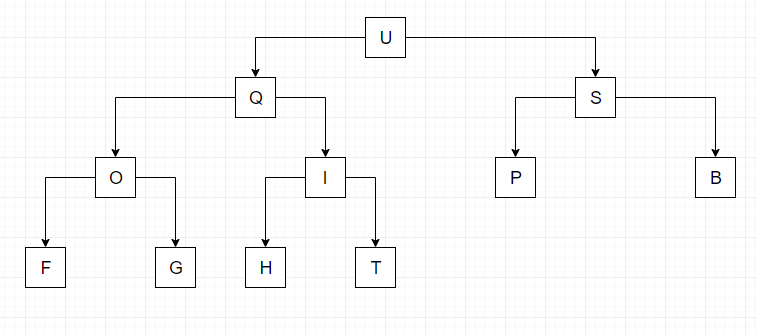
2A. The definition of max heap is for all nodes i, excluding the root, A[PARENT(i)] ≥ A[i]. And the largest element should at root. If we draw the heap for array A.



We could know 6 is greater than 5, and it should be the parent of 5. The correct max heap should be like

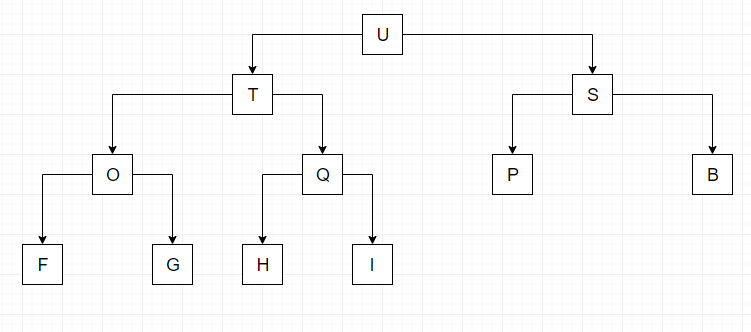


3A. Implement the execution of INSERT(T). We know the insert algorithm is INSERT(X) { Add element X to the end of the max-heap Increment the size of the max-heap by 1 Max-heapify the max-heap //As max-heap condition is probably violated after inserting X }.

First insert element T to the end of max-heap, we have 

Second increase the heap size to 11.

Third Max-heapify the max-heap, which we have

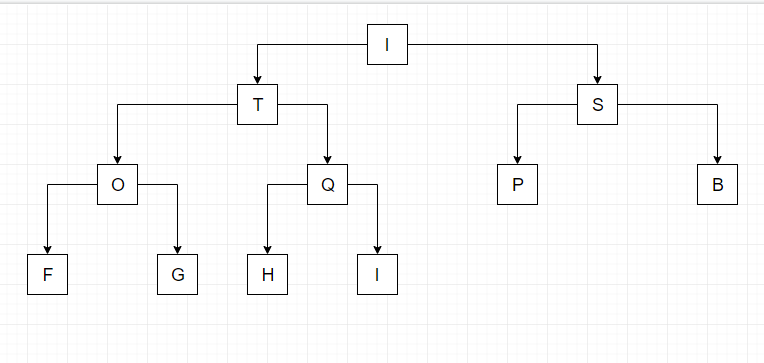


Now Execute EXTRACT-MAX().

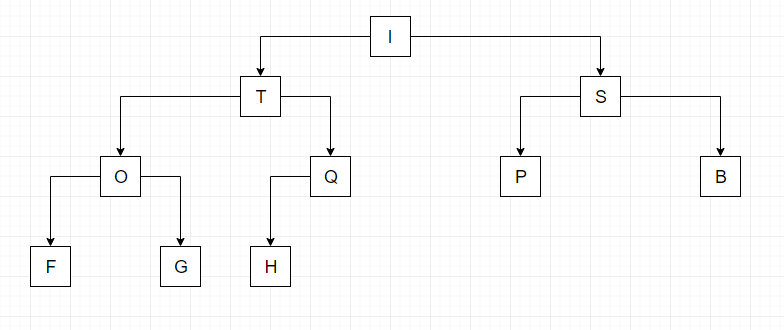
EXTRACT-MAX() { Remove and print the root element from max-heap. Move the last element at the end (leaf) of the heap to the root. Decrement the size of the max-heap by 1 Max-heapify the max-heap //As max-heap condition is probably violated after extraction and moving }

So first print the root element U, and remove it from the heap.

Second move the last element I to the root. We Have

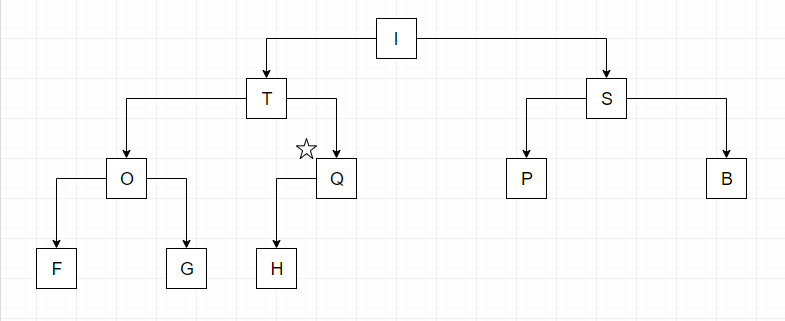


Third decrease the heap size from 11 to 10. We have

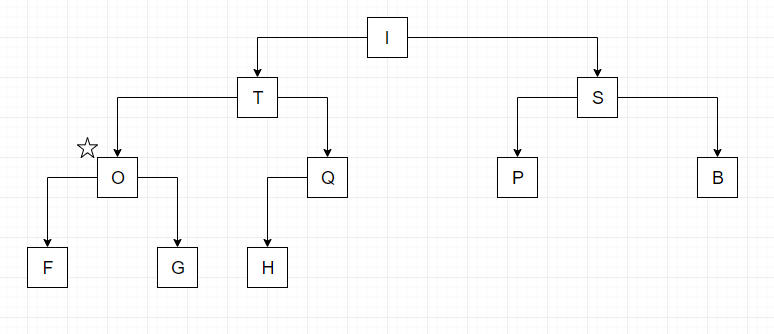


Fourth Max-heapify the max-heap, start from index 5 which is Q, compare to find the max.

Step 1 (Check Q), stay as it is cause Q is max



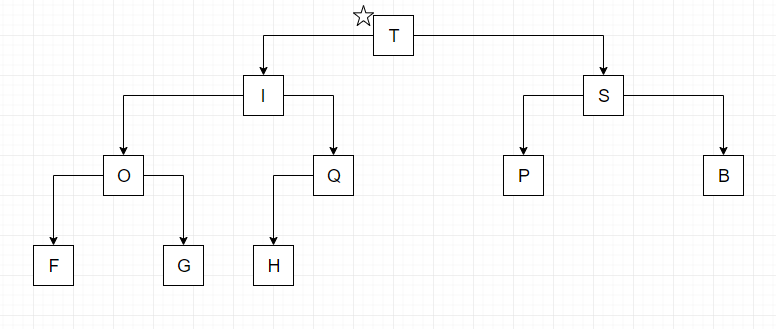
Step 2 (Check O. which is index 4), stay as it is cause O is max

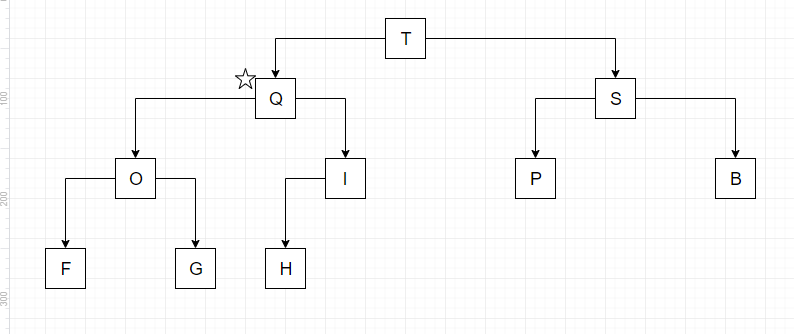


Step 3 (Check S. which is index 3), stay as it is cause S is max.

Step 4 (Check T. which is index 2), stay as it is cause T is max.

Step 5 (Check I. which is index 1), Swap is with T.

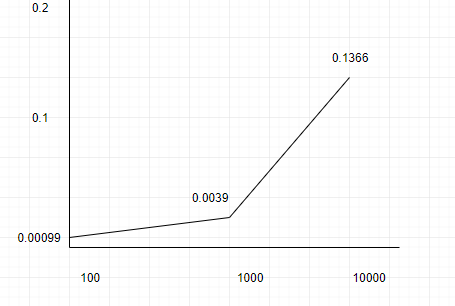


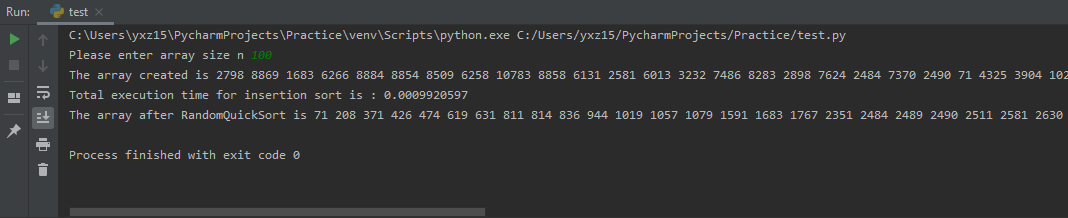
Step 6 (Recursive call the Max-heapify) to Check Q,O,S then I, swap I with Q

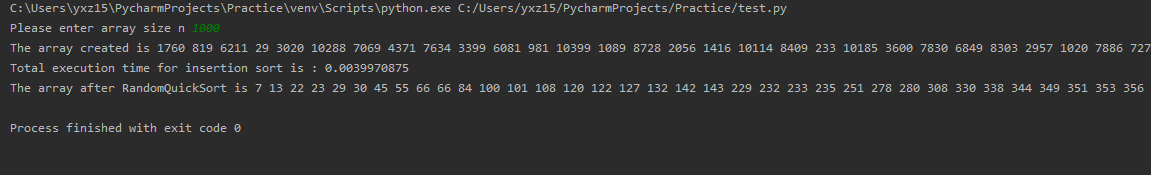
In this case, the operations are done.

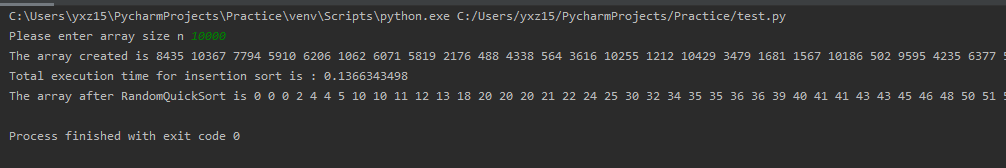
4A.

Plot size vs. time



Output for size 100

Output for size 1000

Output for size 10000

4B. Best case with size 100, which the array is divided by pivot value in exactly two halves and pivot value is the middle.

Example input will be like

92 3 64 76 78 6 4 54 99 87 70 41 58 59 37 29 91 16 74 10 10 10 51 37 11 71 93 5 0 84 61 34 48 68 40 45 25 27 89 56 68 61 74 98 95 88 75 47 27 62 59 53 36 10 26 91 67 62 4 64 3 59 29 92 72 77 100 6 51 11 19 2 7 37 15 92 98 14 88 42 30 67 12 17 72 39 77 22 61 30 20 48 49 40 33 32 96 46 50 27

The execution time for best case quick sort running in my python program is approximately 0.001

4C. Worst case with size 100, which the array is already in ascending or descending order.

Example input will be like

100 98 98 92 91 91 91 90 89 87 87 86 85 83 82 82 79 78 77 77 75 72 71 70 70 69 68 68 67 65 65 65 65 63 63 63 62 62 57 57 56 56 53 51 51 50 50 49 49 49 48 48 47 46 46 45 44 43 43 41 41 41 40 40 39 39 38 38 38 37 36 35 35 34 33 31 31 25 25 25 25 24 23 23 21 19 19 18 16 16 15 14 14 9 9 5 5 1 1 1

The execution time for best case quick sort running in my python program is approximately 0.028

4D I might have some problem when executing the program, because base on the result I saw. Randomly pick the pivot value does shorter the execution time for array size 1000 and 10000,but does not seems for the size 100. But in general, if we have random pivot instead of choosing last element in the array which will prevent array not swapping and choose same pivot every time.