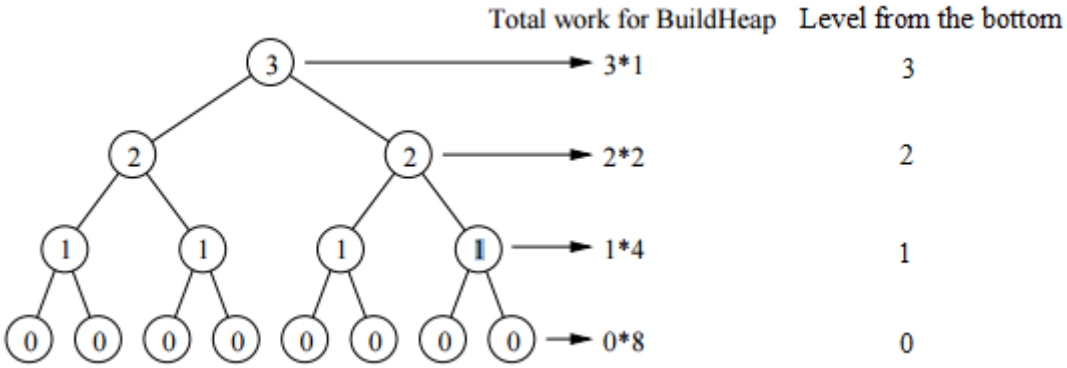
Time Complexity Analysis of Extracting k Largest Numbers in a List Using Heaps
By Niruhan Viswarupan

A. Theoretical Evaluation of the Time Complexity of ExtractMaxK(NumberList, k)

1. Time Complexity Analysis of HeapBuildMaxHeap(A)

Let the number of elements in the input array = n

Assumption: $\exists h \in Z^+$ s.t $n = 2^{h+1} - 1$ i.e the heap is complete



Consider the case where we are at a node in the j^{th} level from the bottom of the heap. Assume that the children of the node are max heaps. The worst case time complexity to max-heapify the node is equal to the number of levels in the subtree rooted at the node if the current key has to sift down all the way to the bottom. That is equal to j.

Number of nodes at the j^{th} level = 2^{h-j}

Total time complexity to max heapify all the nodes at j^{th} level = $j2^{h-j}$

Counting from the bottom to the top of the heap,

Total time taken, T(n) = $\sum_{j=0}^{h} j2^{h-j}$

Now consider,

$\sum_{j=0}^{\infty} x^j = \frac{1}{1-x}$ $|x|<1$

Differentiate both sides w.r.t x,

$\sum_{j=0}^{\infty} jx^{j-1} = \frac{1}{(1-x)^2}$

Multiply by x both sides,

$\sum_{j=0}^{\infty} jx^j = \frac{x}{(1-x)^2}$

Substitute $x = \frac{1}{2}$

$\sum_{j=0}^{\infty} \frac{j}{2^j} = 2$

Now,

$T(n) = \sum_{j=0}^{h} j2^{h-j} = 2^h \sum_{j=0}^{h} \frac{j}{2^j} < 2^h \sum_{j=0}^{\infty} \frac{j}{2^j} = 2^h * 2 = 2^{h+1} = n+1 \qquad \forall h \in Z^+$

$T(n) < n+1$

$\therefore T(n) \in O(n)$

Also clearly $T(n) \in \Omega(n)$ because the algorithm has to access all the elements on the input list at least once.

$\therefore T(n) \in \theta(n)$

2. Cost Modelling of HeapExtractMax(A)

Algorithm:

```
def HeapExtractMax(A):
   last_index = len(A)-1
   (A[1], A[last_index]) = (A[last_index], A[1])
   max_val = A.pop()
   HeapBuildMaxHeap(A)
   return max_val
```

Line	Cost	Times
1	C1	1
2	C2	1
3	C3	1
4	C4	1
5	θ(n)	1
6	C6	1

Time complexity of HeapExtractMax(A), $T1(n) = C1*1 + C2*1 + C3*1 + C4*1 + \theta(n)*1 + C6*1$

$\therefore T1(n) \in \theta(n)$

3. Cost Modelling of ExtractMaxK(NumberList, k)

Algorithm:

```
def ExtractMaxK (NumberList, k):
   output = []
   a = HeapBuildMaxHeap(NumberList)
   for i in range(k):
      output.append(HeapExtractMax(NumberList))
   return output
```

Line	Cost	Times
1	D1	1
2	D2	1
3	θ(n)	1
4	D4	k+1
5	D5+ θ(n)	k
6	D6	1

Time complexity of ExtractMaxK(NumberList, k),
$T2(n, k) = D1*1 + D2*1 + \theta(n)*1 + D4*(k+1) + (D5+ \theta(n))*k + D6*1$
$T2(n, k) \in \theta(kn)$
Since n>>k
$T2(n, k) \in \theta(n)$

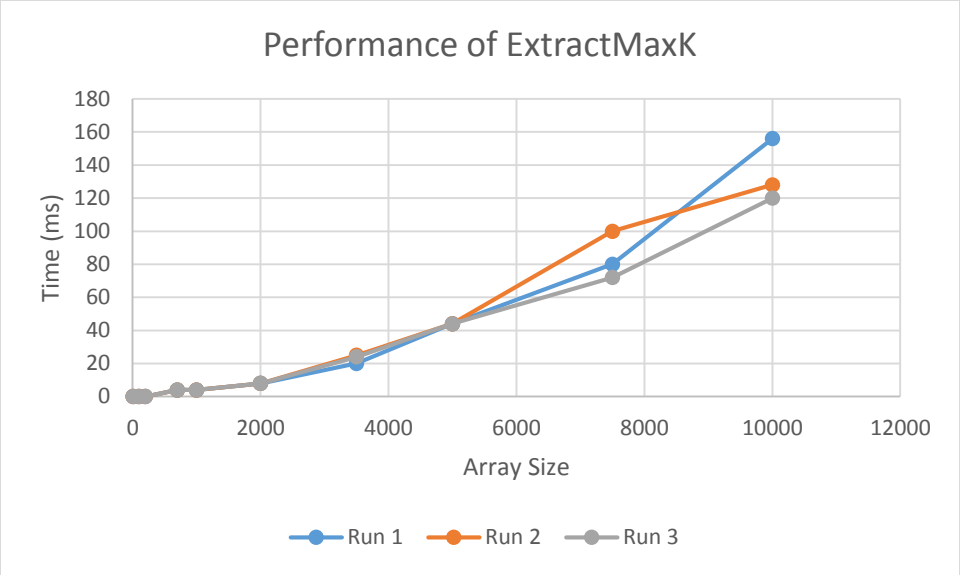
Note:
Here we assume that n>>k. At every iteration of the extract max from the number list the length of the list reduces by one. This happens k times. Since n>>k we neglect the reduction in length of the list during the k iterations.

B. Time Complexity Analysis of ExtractMaxK(NumberList, k) Code Written Using Input

Lists of random integers of the given array sizes were generated and times were measured.
Test Code:

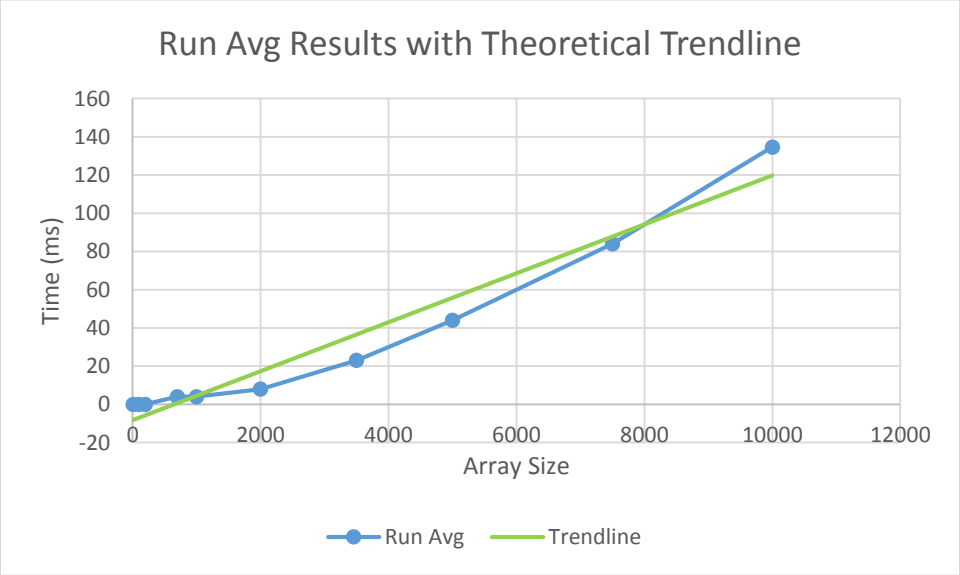
```
def RandListGen(n):
    return [random.randint(0, n) for i in range(n)]

test = [(RandListGen(10), 3), (RandListGen(100), 5), (RandListGen(200), 7), (RandListGen(700), 10),
(RandListGen(1000), 10),
        (RandListGen(2000), 15), (RandListGen(3500), 20), (RandListGen(5000), 30),
(RandListGen(7500), 35), (RandListGen(10000), 40)]

for element in test:
    start = time.time()
    ExtractMaxK(element[0], element[1])
    end = time.time();
    print (end-start)*1000
```

Array Size	Time(ms)			
	Run 1	Run 2	Run 3	Run Avg
10	0	0	0	0
100	0	0	0	0
200	0	0	0	0
700	3.999948502	3.999948502	4.00018692	4.000027974
1000	3.999948502	3.999948502	3.999948502	3.999948502
2000	7.999897003	8.000135422	7.999897003	7.999976476
3500	20.00021935	24.99985695	23.99992943	23.00000191
5000	43.99991035	44.00014877	44.00014877	44.0000693
7500	79.99992371	99.99990463	72.0000267	83.99995168
10000	156.0001373	128.000021	119.9998856	134.6666813



One can see that the time taken for the algorithm increases somewhat linearly with the size of the input. So the time complexity of the ExtractMaxK algorithm can be stated as θ(n).

C. Comparison between Theoretical and Practical Observations of the Time Complexity



It is clear that the run averages follow closely the theoretical prediction of linear time complexity. The possible reasons for deviations are:

1. The distribution of numbers generated by the randint function. This will affect the time taken to build max heaps.
2. The assumptions made during the theoretical derivations
3. Effect of other processes running on the CPU. The time.time() function returns the wall clock time and not the CPU time spent on the process. So other processes running on the CPU could have taken place during the measurement.