Informatics 2 – Introduction to Algorithms and Data Structures

Tutorial 4: The Master Theorem and Heaps

1. In Lecture 8, we gave the following pseudocode for a *binary search* procedure, which searches the dictionary items A[i], A[i+1], ..., A[j−1] within a sorted array A for one matching a given key:

```
binarySearch(A,key,i,j):
    if j−1 = i
        if A[i].key = key
            return A[i].value
        else FAIL
    else
        k = ⌊ i+j/2 ⌋
        if key < A[k].key
            return binarySearch(A,key,i,k)
        else return binarySearch(A,key,k,j)
```

   For $n \geq 1$, let $T(n)$ denote the worst-case number of line executions required to evaluate **binarySearch**(A,key,i,j) when j−i = n.

   (a) Write down an ordinary recurrence relation satisfied by $T(n)$, that is, one that allows us to calculate the exact number of line executions for a given n. (You may make reasonable decisions of your own as to what precisely counts as a 'line execution'.) Justify your recurrence relation as carefully as you can.

   (b) Simplify this down to an *asymptotic* recurrence relation that can be used to determine the growth rate of T. Apply the Master Theorem to find this growth rate. Does this result agree with what you know by other means?

   (c) Also apply the Master Theorem to give tight asymptotic solutions for the following recurrences. Assume $T(1) = \Theta(1)$ in each case, and ignore floor and ceiling issues.

      i. $T(n) = 2T(n/3) + \Theta(n)$
      ii. $T(n) = 7T(n/2) + \Theta(n^2)$
      iii. $T(n) = 2T(n/4) + \Theta(\sqrt{n})$

2. Draw the heap, and each intermediate state, which is created when we apply the Max-Heap-Insert algorithm to the following sequence of elements `12, 5, 4, 8, 9, 1, 16, 20, 7, 6`, starting with an empty heap.

   At each step draw both the tree representation of the heap and the contents of the corresponding array representation.

3. Show that when our input array is n numbers in (non-decreasing) sorted order, then it will take time $\Omega(n \log(n))$ to insert them into an initially empty heap. Give details of the running-time we will have for each of the individual Max-Heap-Insert operations (and why), and then show that the total running-time for this bad case satisfies $\Omega(n \log(n))$.

4. As mentioned in lectures, the implementation of Heaps in Python differs from the traditional set-up, with some operations being slightly different and different names on familiar operations.

   Download a recent release of Python, for example:
   `https://www.python.org/downloads/release/python-386/`

   Once unpacked, look for the file `heapq.py` in the subdirectory `Lib`, and open it (with any text viewer). Go through the file, and make notes on the differences between their implementation and the structure/naming from our lecture notes.