MA214 Algorithms and Data Structures **LT 2021/22**

Solutions 1

Exercise 1.1: The one-dimensional peak problem **1 pt**

The **one-dimensional peak problem**:

Input: A sequence of distinct integers $a_0, a_2, \ldots, a_{n-1}$ such that the numbers are first increasing and then decreasing.

Output: Index i of the peak element. That is, $i = 0$ if $a_0 > a_1$, $i = n - 1$ if $a_{n-2} < a_{n-1}$, and $i = j$ with $0 < j < n - 1$ if $a_{j-1} < a_j$ and $a_j > a_{j+1}$.

Exercise 1.2: A slow iterative algorithm **4 pt**

(a) The algorithm in words:

Suppose the list is called `A` and that it is of length `n`. We go through the elements at position `i = 0, ..., n-2` and return the first index for which `A[i] > A[i+1]` (if any). If no such index exists, then we return `n-1`.

(b) This is the implementation in Python:

```python
def peak1d_iterative(A):
    for i in range(0,len(A)-1):
        if A[i] > A[i+1]:
            return i
    return len(A)-1

A = [2,3,4,10,3,2,1]
print(peak1d_iterative(A))
```

(c) Minimum and maximum number of comparisons:

The minimum number of comparisons is 1, when the first element is the peak, i.e., `A[0] > A[1]`. The maximum number of comparisons is `n-1`, when the last element is the peak, i.e., `A[n-2] < A[n-1]`.

Exercise 1.3: A faster recursive algorithm **5 pt**

(a) This is the algorithm in words:

In addition to the list `A`, which we again assume to be of length `n`, we take the start index `i` and the end index `j` of the subsequence to consider. The base case is when `i == j`, in which case the subsequence is of length 1 and we return `i`. Otherwise, we look at the middle index $\lfloor (\mathtt{i} + \mathtt{j})/2 \rfloor$: or, in Python notation, `m = (i+j)//2`. We compare `A[m]` to `A[m+1]`. If `A[m] < A[m+1]`, then we know that the peak is in `A[m+1],...,A[j]`. So, we recurse with `i` updated to be `m+1`. Otherwise, we know that the peak is in `A[i],...,A[m]`. So, we recurse with `j` updated to be `m`.

(b) This is the implementation in Python:

```python
def peak1d_recursive(A,i,j):
    if i == j:
        return i
    m = (i+j)//2
```

```python
        if A[m+1] > A[m]:
            return peak1d_recursive(A,m+1,j)
        else:
            return peak1d_recursive(A,i,m)

A = [2,3,4,10,3,2,1]
print(peak1d_recursive(A,0,len(A)-1))
```

(c) We can obtain the solution to the recurrence relation as follows.

For $n > 1$, by repeated substitution,

$$\begin{aligned} T(n) &= T(n/2^1) + 1 \\ &= T(n/2^2) + 2 \\ &\vdots \\ &= T(n/2^k) + k. \end{aligned}$$

We want $T(n/2^k)$ to correspond to the base case $T(1)$. For this we need $n/2^k = 1$, which is the case when $k = \log_2(n)$.

Substituting $k = \log_2(n)$ into the above formula gives

$$T(n) = \log_2(n) + 1.$$

Note that this is consistent with the requirement that $T(1) = 1$.

We obtain that $T(n) = \log_2(n) + 1$ for all $n \geq 1$.

Additional questions: If $2^{k-1} < n < 2^k$, then it is acceptable to consider $n' = 2^k$ because then $n' \leq 2n$ and $T(n) \leq T(n') = \log_2(n') + 1 \leq \log_2(2n) + 1 = \log_2(n) + 2$. If, on the other hand, the $+1$ was replaced with a $+3$, then the solution to the recurrence relation would be $3 \log_2(n) + 3$.