Introduction to Algorithms - Reading Notes & Selected Solutions

Nimrod Shneor

October 4, 2019

Chapter 2

Solutions to selected exercises:

2.1-1

$$\begin{aligned}
A &= [31, 41, 59, 26, 41] \\
A &= [31, 41, 59, 26, 41] \\
A &= [31, 41, 26, 59, 41] \\
A &= [31, 26, 41, 59, 41] \\
A &= [26, 31, 41, 59, 41] \\
A &= [26, 31, 41, 41, 59]
\end{aligned}$$

2.1-4

Algorithm 1 BinaryAddition(A,B,n)

```
carry ← 0
for i ← 1 to n: do
    C[i] ← (A[i] + B[i] + C[i]) mod2
    carry ← A[i] * B[i]
end for
C[i + 1] ← carry
return C
```

- Input: two n-bit numbers A, B.
- Output: the sum of A, B - an $n + 1$-bit number.

2.2-3 Define X = The number of elements checked in a "brute force" linear search.

Than $X \in \{1...n\}$ and the average number of elements checked in a linear search is exactly:

$$E[X] = \sum_{i=1}^{n} \frac{1}{n} i = \frac{1}{n} \sum_{i=1}^{n} i = \frac{n(n-1)}{2n} = \Theta(n)$$

The worst case is where the last element of the array is the one searched for - resulting in an $\Theta(n)$ run time.

2.3-3

$$T(n) = \begin{cases} 2 & n = 2 \\ 2T(\frac{n}{2}) + n & n > 2 \end{cases}$$

Q: Proof by induction that if n is an exact power of two (that is $n = 2^k$ for some constant $k \geq 1$) than $T(n) = nlogn$.

Proof: By induction.

Base case: for $k = 1$ than $T(n) = 2 = 2log2 = nlogn$
Assumption: Assume the above holds for all integers up to $k > 1$.
Induction step: We now prove the statement for $n = 2^{k+1}$.
Plugging in to the formula

$$\begin{aligned} T(n) = 2T(\frac{n}{2}) + n = 2T(\frac{2^{k+1}}{2}) + 2^{k+1} \\ = 2T(2^k) + 2^{k+1} = 2 * 2^k log 2^k + 2^{k+1} \\ = k2^{k+1} + 2^{k+1} = (k+1)2^{k+1} = 2^{k+1} log 2^{k+1} \\ = nlogn \end{aligned}$$

□

2.3-5

Algorithm 2 BinarySearch(A,x)

$l \leftarrow 0$
$r \leftarrow length(A)$
while $l < r - 1$ **do**
  if $x = A[\frac{l+r}{2}]$ **then**
    $\frac{l+r}{2}$
  end if
  if $x > A[\frac{l+r}{2}]$ **then**
    $l = A[\frac{l+r}{2}]$
  else
    $r = A[\frac{l+r}{2}]$
  end if
end while
return -1

At each iteration of the while loop the distance between the two pointers - l, r - is halfed until the element is found or we return -1. The while loop will terminate once the two pointers are at distance two at which point either the element x is found or the loop will terminate. Thus the distance between the two pointers at each iteration i is percisly $\frac{length(A)}{2^i} = \frac{n}{2^i}$

At the time of the termination the distance between the two pointers is two, thus -

$$\begin{aligned} \frac{n}{2^i} &= 2 \\ n &= 2^{i+1} \\ log(n) &= i + 1 \\ log(n) - 1 &= i \\ \Theta(log(n)) &= i \end{aligned}$$

Algorithm 3 FindSum(A,x)

2.3-7

$B \leftarrow MergeSort(A)$
$l \leftarrow 0$
$r \leftarrow length(B)$
while $l < r$ **do**
  if $B[l] + B[r] == x$ **then**
    return true
  else if $B[l] + B[r] < x$ **then**
    $l \leftarrow l + 1$
  else
    $r \leftarrow r - 1$
  end if
end while
return false

Solutions to selected problems:

2-1

a Given $\frac{n}{k}$ lists each of size k. Applying $InsertionSort$ to each list seperatly yealds worst-case runtime of $\Theta(k^2)$. Doing this for all $\frac{n}{k}$ lists yealds an $\Theta(\frac{n}{k}k^2) = \Theta(nk)$ runtime algorithm.

b

Algorithm 4 ModifiedMergeSort(A)

Split A to form $S = [A_1, .., A_{\frac{n}{k}}]$ array of sub-arrays of size k
for $i \leftarrow 1$ to $\frac{n}{k}$ **do**
  $A_i \leftarrow InsertionSort(A_i)$
end for
while $|S| > 1$ **do**
  $l \leftarrow 1$
  $r \leftarrow length(S)$
  $S' \leftarrow \Phi$
  while $l < r$ **do**
    $S' \leftarrow S' \bigcup Merge(A_l, A_r)$
    $l \leftarrow l + 1$
    $r \leftarrow r - 1$
  end while
  $S \leftarrow S'$
end while

We prove that at each iteration of the outer while loop the size of $|S|$ is $\frac{n}{2^i k}$.
Proof: By induction,

Base $i = 1$: In the first iteration we set l and r to hold the two opposit ends of S, at each iteration we merge two subsets and continue so on until $l = r$ or $l > r$ (depending on the number of subsets) because at each iteration we merged two subsets the number of iterations of the inner loop is percisly $\frac{n}{2k}$.

Step: Assume that the number of subsets in $|S|$ is $\frac{n}{2^i k}$ at iteration i next we prove that at iteration $i + 1$ the above statement holds.

Again, from the same argument for the base case - at each iteration of the inner loop the number of elements decrease by two the number of iterations of the inner loop is $\frac{n}{2^{i+1}k}$ yeilding that number of subsets.

The outer loop will terminate once $|S| = 1$, that is -

$$\frac{n}{2^i k} = 1$$
$$\frac{n}{k} = 2^i$$
$$log(\frac{n}{k}) = i$$

At each iteration we perform $\frac{n}{2^i k}$ merges each runs in $\Theta(2^i k)$ for a total of $\Theta(n)$, Thus the total running time of the while loop is $\Theta(nlog(\frac{n}{k}))$

All together we get $\Theta(nlog(\frac{n}{k})) + \Theta(nk)$.