COMP 3711
2024

Notes by Marcus Chan

March 10, 2024

Contents

1 Tutorials

1.1 Tutorial 1 - Asymptotic notation

1. (a) true
   (b) false
   (c) true
   (d) false
   (e) false
   (f) false
   (g) true
2. (a) true
   (b) false
   (c) false
3. (a) yes
   (b) no.
   let $T_i(n) = i \cdot n$ and $f(n) = n$

$$\begin{aligned} g(n) &= \frac{n(n+1)}{2} \cdot f(n) \\ &= O(n^2 f(n)) \\ &= O(n^3). \end{aligned}$$

$$\therefore g(n) \neq O(f(n)) \text{and } \neq O(n(f(n)).$$

   note: this is because

$$\sum_{i=1}^{n} O(f(n)) \neq O(\sum_{i=1}^{n} f(n)).$$

1.2 Tutorial 2 - Divide and conquer

1. done
2. done
3. find the median. If k > median, then remove all i from A s.t where A[i] < median, vice versa. Time complexity: $\frac{n}{2} + \frac{n}{4} + \ldots + \frac{n}{n} = \frac{(\frac{1}{2})^{\log_2 \frac{n}{2}} - 1}{\frac{1}{2} - 1} \cdot \frac{n}{2} = \frac{\frac{1}{\frac{n}{2}} - 1}{\frac{1}{2} - 1} \cdot \frac{n}{2} = \frac{1 - \frac{n}{2}}{\frac{1}{2} - 1} = O(n)$
4. find the median x_k and then $w = \sum_{i=k}^{n} w_k$. W.L.O.G, if $w > \frac{1}{2}$, then our solution $j > k$. Remove $A[i]$ such that $i < k$. Set $w_k = w_k + 1 - w$. If $w < \frac{1}{2}$, remove all $A[j]$ where $j > k$. Set $w_k = w$. Continue the recursion. The time complexity will be $O(n)$ (analysis the same as Q3)

1.3 Tutorial 3 - Divide and conquer

1. done
2. take a pivot. check which side is larger. Take the larger side. If all elements in one side is the same and len $> \frac{n}{2}$ then we have our answer. Recurse until both sides have len $< n/2$
3. example: 5, -10, -10, 0, 60

1.4 Tutorial 4 - Randomized algorithm

1. (a) probability that you hire exactly one person

$$\begin{aligned} P(\texttt{one person}) &= \sum_{i=1}^{n} \frac{1}{i} \\ &= O(n \log n) \end{aligned}$$

.

correct answer: $\frac{1}{n}$

(b) probability that you hire exactly n person

$$\begin{aligned} P(\texttt{n person}) &= \prod_{i=1}^{n} \frac{1}{i} \\ &= \frac{1}{n!} \end{aligned}$$

.

2. Let $x_i = 1$ when the hat is correct

$$\begin{aligned} E[x_i] &= P[x_i] \\ &= \frac{n-i}{n} && \texttt{correct answer: } \frac{1}{n} \\ E[x] &= \sum_{i=1}^{n} E[x_i] \\ &= O(n \log n) && \texttt{correct answer: } 1 \end{aligned}$$

.

3. Let $x_{i,j} = 1$ when $A[i] > A[j]$

$$\begin{aligned} E[x_{i,j}] &= \frac{1}{2} \\ E[x] &= \frac{\binom{n}{2}}{2} \\ &= \frac{n(n-1)}{4} \end{aligned}$$

.

1.5 Tutorial 5 - Selection and quick sort

1. (a) when z_k is in between z_i and z_j, a
   (b) i and j are on the left side of the decision tree, the number of elements $= k - j + 1$. So the probability that z_i or z_j is selected is $Pr[X_{ij} = 1] = \frac{2}{j-k+1}$
   (c) i and j are on the right side of the decision tree, the number of elements $= k - j + 1$. So the probability that z_i or z_j is selected is $Pr[X_{ij} = 1] = \frac{2}{j-k+1}$

2 Lectures

2.1 Lecture 1 - Asymptotic Notation

Theorem 2.1

Upper bounds $T(n) = O(f(n))$.
if exists constants $c > 0$ and n_0 such that $\forall n \geq n_0$, $T(n) \leq c \cdot f(n)$

Theorem 2.2

Lower bounds $T(n) = \Omega(f(n))$
if exists constants $c > 0$ and n_0 such that $\forall n \geq n_0$, $T(n) \geq c \cdot f(n)$

Theorem 2.3

Tight bounds $T(n) = \Theta(f(n))$
if $T(n) = O(f(n))$ and $T(n) = \Omega(f(n))$

constant $9999^{9999^{9999}}$ $<$ logarithmic $\log^{10} n <$ polynomial $n^{0.1} < n \log n < n^2$ $<$ exponential 2^n

1. $9999^{99999^{9999}} = \Theta(1) = O(log(log(n)))$
2. $log(log(n)) = O(logn)$
3. $n^{100} = O(2^n)$ Let $n^{100} = c \cdot 2^n$ s.t. for $n \geq n_0 n^{100} \leq c \cdot 2^n$

$$
\begin{aligned}
100 * \log(n) &= c \cdot n \log(2) \\
n &= \frac{100}{c} * \log(n) \\
&.
\end{aligned}
$$

$\therefore \forall c, n^{100} = O(2^n)$

Theorem 2.4

Common expressions

1. $max(f(n), g(n)) = \Theta(f(n) + g(n))$
2. $\log \sqrt{n} = \Omega(\sqrt{\log(n)})$
3. $\log(2^n) = \Theta(\log(3^n))$
4. $\sum_{i=1}^{n} \frac{1}{i} = \Theta(\log n)$

2.2 Lecture 2 - Running time of sorting

2.2.1 Selection sort

Algorithm 1 Selection sort

```
for i ← 1 to n do
    for j ← i + 1 to n do
        if A[j] < A[i] then
            swap A[j], A[i]
        end if
    end for
end for
```

number of comparisons : $\sum_{i=1}^{n}(n-i) = \sum_{i=1}^{n-1} = \frac{n(n-1)}{2}$

Theorem 2.5

Correctness of selection sort

Prove by induction. Assume the algorithnm sorts every array of size n-1 correctly.

1. It first pusts the smallest item in $A[1]$
2. then runs selection sort on $A[2 \dots n]$ (by induction this is correct)
3. since $A[1]$ is smaller than every other items, the array is sorted

2.2.2 Insertion sort

Algorithm 2 Insertion sort

```
for i ← 2 to n do
    j ← i − 1
    while j ≥ 1 and A[j] > A[j + 1] do
        swap A[j] and A[j + 1]
        j ← j − 1
    end while
end for
```

1. number of comparisons : $\sum_{i=1}^{n}(i-1) = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2}$
2. average case analysis: only half of the keys are compared
3. best case: on sorted data, takes $O(n)$ time

Theorem 2.6

Comparison of running time

1. $O(\log n)$ U $\Theta(2^{\log_2 \log_2 n})$
2. $O(n^4)$ U $O(n^3)$

Exercise 2.7

Prove that $\log(n!) = \Theta(n \log n)$

1. first, prove $\log(n!) = O(n \log n)$

$$\begin{aligned}
\log(n!) &= \sum_{i=1}^{n} \log(i) \\
&\leq \sum_{i=1}^{n} \log(n) \\
&= O(n \log n).
\end{aligned}$$

2. then, prove $\log(n!) = \Omega(n \log n)$

$$\begin{aligned}
\log(n!) &= \sum_{i=1}^{n} \log(i) \\
&\geq \sum_{i=\frac{n}{2}}^{n} \log(i) \\
&\geq \sum_{i=\frac{n}{2}}^{n} \log(\frac{n}{2}) \\
&= \frac{n}{2} \log \frac{n}{2} \\
&= \frac{n}{2}(\log n - \log 2) \\
&= \Omega(n \log n).
\end{aligned}$$

$\therefore \log(n!) = \Theta(n \log n)$

2.3 Lecture 3 - Divide and conquer

Theorem 2.8

Run time analysis of Binary search

$$
\begin{aligned}
T(n) &= T(\frac{n}{2}) + 2 \text{ if } n > 1, \text{ with } T(1) = 1 \\
&= T(\frac{n}{2^2} + 2) + 2 \\
&= \dots \\
&= T(\frac{n}{2^{\log_2 n}}) + 2\log_2 n \\
&= 1 + 2\log_2 n.
\end{aligned}
$$

Examples

1. Rotated sorted array
2. Find the last 0

Algorithm 3 Tower of hanoi(n, peg1, peg2, peg3)

```
if n=0 then return
else
    Tower of hanoi(n-1, peg1, peg2, peg3)        ▷ T(n-1)
    move the only disc from peg 1 to peg 3       ▷ T(1)
    Tower of hanoi(n-1, peg2, peg1, peg3)        ▷ T(n-1)
end if
```

Theorem 2.9

Recurrence of Tower of hanoi:

$$
\begin{aligned}
T(n) &= 2T(n-1) + 1 \\
&= 2(2T(n-2) + 1) + 1 \\
&= \dots \\
&= 2^{n-1}T(1) + 2^{n-2} + 2^{n-3} + \dots + 2^2 + 2 + 1 \\
&= 2^n - 1.
\end{aligned}
$$

Algorithm 4 Merge sort(A, l, r)

if $l = r$ **then** `return`
else
  mid $\leftarrow \frac{l+r}{2}$
  `Merge Sort(A, l, mid)` ▷ T($\frac{n}{2}$)
  `Merge Sort(A, mid+1, r)` ▷ T($\frac{n}{2}$)
  `Merge(A, l, mid, r)` C;ommentO(n)
end if

Theorem 2.10

Analysis of merge sort

$$
\begin{aligned}
T(n) &\leq T(\left\lfloor \frac{n}{2} \right\rfloor) + T(\left\lceil \frac{n}{2} \right\rceil) + O(n) \\
&= 2^i T(\frac{n}{2^i}) + in \qquad \text{where } i = \log_2 n \\
&= n \log_2 n + n.
\end{aligned}
$$

2.3.1 Inversion number

Definition 2.11. Inversion number

Given an array $A[1..n]$, two elements $A[i]$ and $A[j]$ are inverted if $i < j$ but $A[i] > A[j]$

Algorithm 5 Count number of inversion

Input: A, p, q, r
Output: c

L $\leftarrow A[p \dots q]$ and R $\leftarrow A[q \dots r]$
c $gets0$
while $i \leftarrow 0 \leq p - q + 1$ and $j \leq r - q$ **do**
  if $L[i] \leq R[i]$ **then**
    $i \leftarrow i + 1$
  else
    $I[j] = q - p - i + 2$
    $c \leftarrow c + I[j]$
    $j \leftarrow j + 1$
  end if
end while
return *state*

Reccurence $T(n) = 2T(\frac{n}{2}) + n$

Algorithm 6 Sort and count

Input: A, p, r
Output: c

if $p = r$ **then**
    return 0
end if
$c_1 \leftarrow$ SORT-AND-COUNT(A, p, q)
$c_2 \leftarrow$ SORT-AND-COUNT$(A, q+1, r)$
$c_3 \leftarrow$ MERGE-AND-COUNT(A, p, q, r)
return $c_1 + c_2 + c_3$

Algorithm 7 Merge and count

Input: A, p, q, r
Output: c

L $\leftarrow A[p \dots q]$ and R $\leftarrow A[q \dots r]$
c $\leftarrow 0$
for $k \leftarrow p$ to r **do**
    if $L[i] \leq R[i]$ **then**
        $A[k] \leftarrow L[i]$
        $i \leftarrow i + 1$
    else
        $A[k] \leftarrow R[j]$
        $I[j] = q - p - i + 2$
        $c \leftarrow c + I[j]$
        $j \leftarrow j + 1$
    end if
end for
return c

2.3.2 Subarray

1. Maximum subarray (DC v.s. Kadane's algorithm)

Algorithm 8 Kadane's algorithm

Input: *input*
Output: *output*

$V \leftarrow 0$
$maxi \leftarrow -\infty$
for $i \leftarrow 1$ to n **do**
  $V \leftarrow V + A[i]$
  if $V < A[i]$ **then**
    $V = Ai]$
  end if
  if $V > maxi$ **then**
    $maxi = Ai]$
  end if
end for
return *state*

2.3.3 Integer multiplication

Theorem 2.12

Karatsuba Multiplication

Let $a = a_1 2^{\frac{n}{2}} + a_0$ and $b = b_1 2^{\frac{n}{2}} + b_0$

$$ab = a_1 b_1 2^n + (a_1 b_0 + a_0 b_1) 2^{\frac{n}{2}} + a_0 b_0$$
$$a_1 b_0 + a_0 b_1 = (a_0 + a_1) \cdot (b_0 + b_1) - a_0 b_0 - a_1 b_1 \quad \text{(reduce to 3 multiplications)}$$
$$.$$

$\implies T(n) = 3T(\frac{n}{2}) + n$

2.4 Lecture 4 - Master theorem

2.4.1 intro

$T(n) = aT(\frac{n}{b}) + f(n)$

1. $f(n) = n^{c+\epsilon}$ then $T(n) = \Theta(n^c)$
2. $f(n) = n^c$ then $T(n) = O(n^c)$
3. $f(n) = n^{c-\epsilon}$, then $T(n) = \Omega(f(n)))$

2.4.2 inequality

$T(n) = aT(\frac{n}{b}) + f(n)$

1. $f(n) = O(n^{c+\epsilon})$ then $T(n) = O(n^c)$

2. $f(n) = O(n^c)$ then $T(n) = O(n^c)$
3. $f(n) = O(n^{c-\epsilon})$, then $T(n) = O(f(n))$

Remark 2.13 — when $f(n) = \Theta(n)$, we will split the cases by checking c > 1, c $= 1$ and c < 1

Proof when $f(n) = O(n)$

1.

$$\begin{aligned}
T(n) &\leq aT(\frac{n}{b}) + f(n) \\
&= a \cdot (T(\frac{n}{b^2}) + \cdot(\frac{n}{b})) + f(n) \\
&= \ldots \\
&= a^i T(1) + a^{i-1}(\frac{n}{b^i}) + \ldots + kn \\
&= n^c + kn \sum_{i=0}^{\log_b(n)-1} (\frac{a}{b})^i && \texttt{case 1:} \ c > 1 \\
&= n^c + kn \cdot \frac{(\frac{a}{b})^{\log_b n} - 1}{\frac{a}{b} - 1} \\
&\leq n^c + kn \cdot \frac{(\frac{a}{b})^{\log_b n}}{\frac{a}{b} - 1} \\
&= n^c + k \cdot \frac{n^{\log_b a}}{\frac{a}{b} - 1} \\
&= O(n^c)
\end{aligned}$$

.

2.

$$\begin{aligned}
T(n) &\leq aT(\frac{n}{b}) + f(n) \\
&= n + kn \sum_{i=0}^{\log_b(n)-1} (\frac{a}{b})^i && \texttt{case 2:} \ c = 1 \\
&= n + kn \cdot (\log_b n - 1) \\
&= O(n \log n)
\end{aligned}$$

.

3.

$$
\begin{aligned}
T(n) &\leq aT(\frac{n}{b}) + f(n) \\
&= n^c + kn \sum_{i=0}^{\log_b(n)-1} (\frac{a}{b})^i && \texttt{case 3:}\ c < 1 \\
&= n^c + kn \cdot O(1) && \texttt{decreasing geometric series} \\
&= O(n)
\end{aligned}
$$

.

2.5 Lecture 5 - Randomized algorithms

Exercise 2.14

$$
\begin{aligned}
S = \sum_{i=1}^{\infty} i(1-p)^{i-1} &= \sum_{j=0}^{\infty} (j+1)(k)^j \\
kS &= \sum_{j=1}^{\infty} (j+1)(k)^j && (1) \\
S - kS &= 1 + k + k^2 + k^3 + \ldots = \frac{1}{1-k} && (2) \\
S &= \frac{1}{(1-k)^2}
\end{aligned}
$$

.

Exercise 2.15 (Coupon collector)

$$
\begin{aligned}
P(i) &= \frac{n-i}{n} \\
E[x_i] &= \frac{1}{P(i)} \\
E[X] &= \sum_{i=1}^{n-1} E[x_i] \\
&= n \sum_{i=1}^{n} \frac{1}{i} \\
&= \Theta(n \log n)
\end{aligned}
$$

.

1. hiring problem
2. birthday paradox
3. generate random permutation (with proof)

2.6 Lecture 6 - Quick Sort

2.6.1 Intro to algo

Algorithm 9 Quicksort

Input: A, p, q, r
Output: 1

if $p \geq r$ **then**
    `return`
end if
$q =$ PARTITION(A, p, r)
QUICKSORT$(A, p, q - 1)$
QUICKSORT$(A, q + 1, r)$
return *state*

Algorithm 10 Partition

Input: A, p, r
Output: q

$i \leftarrow p - 1$
for $j \leftarrow p$ to $r - 1$ **do**
    if $A[j] \leq x$ **then**
        $i \leftarrow i + 1$
        SWAP$(A[i], A[j])$
    end if
end for
return i

2.6.2 Run time analysis

1. best case: $O(\log n)$
2. worse case: $O(n^2)$

Average run-time of quicksort. To attain average case, randomly select the pivot. Let $p(x_{i,j})$ be the probability that z_i and z_j are compared

$$
\begin{aligned}
p(x_{i,j}) &= \frac{1}{j-i+1} \\
E[X] &= \sum_{i=1}^{n} 2 \cdot p(x_{i,j}) && z_i \texttt{ and } z_j \texttt{ are symmetric} \\
&= O(n \log n)
\end{aligned}
$$

.

Exercise 2.16 (k-smallest element in union of two sorted lists)
binary search

2.7 Lecture 7 - Heapsort

priority queue

1. insert min in $O(\log n)$
2. extract min in $O(\log n)$

Definition 2.17. min-heap

1. children is as large as the parent
2. add new element to the next available position at the lowest level

Definition 2.18. heap-sort
build a binary heap and insert elements, extract min in $O(n \log n)$

Remark 2.19 — How to support decrease-key in $O(\log n)$?

2.8 Lecture 8 - Sorting in linear time

Claim 2.20. Properties of sorting

1. Comparison based sorting have $\Omega(n \log n)$
2. $h \geq \log n$

Algorithm 11 Extract min

Input: A
Output: min

$min \leftarrow A[i]$
ASSIGN(A[1],A[n])
ASSIGN(A[n],∞)
$j \leftarrow 1$
$l \leftarrow A[2j]; r \leftarrow A[2j+1]$
while $A[j] > min(A[l], A[r])$ **do** ▷ heapify
    if $l > r$ **then**
        SWAP(A[j],A[2j])
        $j \leftarrow 2j$
    else
        SWAP(A[j],A[2j+1])
        $j \leftarrow 2j+1$
    end if
    $l \leftarrow A[2j]; r \leftarrow A[2j+1]$
    return min

2.8.1 Counting sort

Algorithm 12 Counting sort

Input: A, k
Output: B

$C[0 \dots k] \leftarrow 0$
for $i \leftarrow 1$ to n **do**
    $C[A[i]] \leftarrow C[A[i]] + 1$
end for
for $i \leftarrow 1$ to k **do**
    $C[i] \leftarrow C[i] + C[i-1]$
end for
for $i \leftarrow n$ to 1 **do**
    $idx \leftarrow C[A[i]]$
    $B[idx] \leftarrow A[i]$
    $C[A[i]] \leftarrow C[A[i]] - 1$
end for
return B =0

2.8.2 Radix sort

Definition 2.21 (Radix sort)**.** use counting sort to sort digits from least significant to most significant

Proof of correctness.

Assume we already sorted the lower digits $0 \ldots k-1$. We need to sort the most significant digit. WLOG, suppose we have a digit d, by definition, all numbers with d as the k-th digit are sorted but not necessarily in continuous positions. Counting sort simply puts numbers with the same k-th digit in the continuous order.

Run-time analysis.

Counting sort is run d times and total run time is $\Theta(d(n+k))$

Decision model on list merging.

1. number of leafs $= k!$
2. number of possible permutations $= (k!)^{\frac{n}{k}}$
3. height $= \log((k!)^{\frac{n}{k}}) = \Theta(n \log k)$