0.1 Lecture 1 - Asymptotic Notation

Theorem 0.1

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 0.2

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 0.3

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 $n \geq n_0 n^{100} \leq c \cdot 2^n$

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

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

Theorem 0.4

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. $\sum_{i=1}^{n} \frac{1}{i} = \Theta(\log n)$

0.2 Lecture 2 - Running time of sorting

0.2.1 Selection sort

Algorithm 1 Selection sort

for $i \leftarrow 1$ to n **do**
  for $j \leftarrow i+1$ 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 0.5

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 $A[2 \ldots n]$ (by induction this is correct)
3. since $A[1]$ is smaller than every other items, the array is sorted

0.2.2 Insertion sort

Algorithm 2 Insertion sort

for $i \leftarrow 2$ to n **do**
  $j \leftarrow i-1$
  while $j \geq 1$ and $A[j] > A[j+1]$ **do**
    `swap` $A[j]$ `and` $A[j+1]$
    $j \leftarrow j-1$
  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 0.6

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 0.7

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 $\log(n!) = \Omega(n \log n)$

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

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

0.3 Lecture 3 - Divide and conquer

Theorem 0.8

Run time analysis of Binary search

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

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 0.9

Recurrence of Tower of hanoi:

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

Algorithm 4 Merge sort(A, l, r)

```
if l = r then return
else
    mid ← (l+r)/2
    Merge Sort(A, l, mid)                 ▷ T(n/2)
    Merge Sort(A, mid+1, r)               ▷ T(n/2)
    Merge(A, l, mid, r) C;ommentO(n)
end if
```

Theorem 0.10

Analysis of merge sort

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

0.3.1 Inversion number

Definition 0.11. 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 ← A[p...q] and R ← A[q...r]
c  gets0
while i ← 0 ≤ p − q + 1 and j ≤ r − q do
    if L[i] ≤ R[i] then
        i ← i + 1
    else
        I[j] = q − p − i + 2
        c ← c + I[j]
        j ← 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 $\leftarrow A[p \ldots q]$ and R $\leftarrow A[q \ldots r]$
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

0.3.2 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*

0.3.3 Integer multiplication

Theorem 0.12

Karatsuba Multiplication

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

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

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