

CPSC 311-501 Homework 1

- **Ex. 1.2-2. Justify your answer.**

Find values of n such that

$$8n^2 < 64 n \lg n.$$

$$n < 8 \lg n$$

$$n = 2, 3, \dots, 43$$

- **Ex. 2.2-3. Also analyze the best case.**

- *How many elements of the input sequence need to be checked on the average, assuming that the element being searched for is equally likely to be any element in the array?*

$$\frac{1}{n}(1 + 2 + \dots + n) = \frac{1}{n} \left(\frac{n(n+1)}{2} \right) = \frac{(n+1)}{2} \text{ elements}$$

- *How about in the worst case?*

n elements

- *What are the average-case and worst-case running times of linear search in Θ -notation?*

average-case: $\Theta(n)$

worst-case: $\Theta(n)$

- *The best case?*

1 element (i.e., the first element in the array) needs to be checked. The best-case running time of linear search in Θ -notation is $\Theta(1)$.

- **Ex. 2.3-3**

(i) When $n = 2$, $T(n) = 2 = 2 \lg 2$.

(ii) Assume that $T(n) = n \lg n$ for $n = 2^k$, $k > 1$.

$$\begin{aligned} \text{Then } T(2n) &= 2T(2n/2) + 2n \\ &= 2T(n) + 2n \\ &= 2(n \lg n) + 2n \\ &= 2n(\lg n + 1) \\ &= 2n(\lg n + \lg 2) \\ &= 2n \lg(2n) \end{aligned}$$

- **Ex. 2.3-5**

- *Write pseudocode, either iterative or recursive, for binary search.*

Recursive_Binary_Search(A , first, last, v)

```
{
    if (first < last)
        return NIL;
    half = floor((first + last) / 2);
    if (v == A[half])
        return half;
    else if (v < A[half])
        RECURSIVE_BINARY_SEARCH(A, first, half, v);
    else
        RECURSIVE_BINARY_SEARCH(A, half + 1, last, v);
}
```

- *Argue that the worst-case running time of binary search is $\Theta(\lg n)$.*

The recurrence for the worst-case running time of Recursive_Binary_Search is: $T(n) = T(n/2) + \Theta(1)$. $T(n) =$

$\Theta(n^{\log_2 1} \lg n) = \Theta(n^0 \lg n) = \Theta(\lg n)$ by the Master theorem – case 2.

• **Ex. 2.3-6**

No. The cost of moving array values to the right in the while loop is still $\Theta(n)$. Thus, the overall worst-case running time of insertion sort using a binary search is $\Theta(n^2)$

• **Prob. 2-4**

▪ (a)

(1,5) (2,5) (3,4) (3,5) (4,5)

▪ (b)

□ What array with elements from the set $\{1, 2, \dots, n\}$ has the most inversions?

$\langle n, n-1, \dots, 1 \rangle$

□ How many does it have?

$$(n-1) + (n-2) + \dots + 1 = \frac{(n-1)n}{2}$$

▪ (c)

The while loop in the INSERTION-SORT procedure (p. 24 in the textbook) is executed when $i < j$ and $A[i] > A[j]$. Thus, the number of inversions in the input array, NI, is equal to the number of times the body of the while loop is executed for $2 \leq j \leq \text{length}[A]$. That is, $NI = \sum_{j=2}^n (t_j - 1)$. Using this equality, we can express $T(n)$ in p. 24 as $T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5(NI + (n-1)) + c_6NI + c_7NI + c_8(n-1)$.

▪ (d)

IMERGE (A, p, q, r) //Adapted from the MERGE procedure in p. 29.

```

1   n1 ← q-p+1
2   n2 ← r-q
3   create arrays L[1 .. n1+1] and R[1 .. n2+1]
4   for i ← 1 to n1
5       do L[i] ← A[p+i-1]
6   for j ← 1 to n2
7       do R[j] ← A[q+j]
8   L[n1+1] ← ∞
9   R[n2+1] ← ∞
10  i ← 1
11  j ← 1
12  num_inversions ← 0
13  for k ← p to r
14      do if L[i] ≤ R[j]
15          then A[k] ← L[i]
16              i ← i+1
17          else A[k] ← R[j]
18              j ← j+1
19              num_inversions ← num_inversions + n1 + 1 - i
20  return num_inversions

```

IMERGE-COUNT (A, p, r) //Adapted from the MERGE-SORT procedure in p. 32.

```

1  num_inversions ← 0
2  if p < r
3      then q ← ⌊(p+r)/2⌋
4          num_inversions ← num_inversions + IMERGE-COUNT (A, p, q)
5          num_inversions ← num_inversions + IMERGE-COUNT (A, q+1, r)

```

```

6      num_inversions ← num_inversions + IMERGE (A, p, q, r)
7      print num_inversions

```

• **Ex. 3.1-4. Justify your answers mathematically according to the formal definition of big-oh.**

- $Is 2^{n+1} = O(2^n)$?

Yes. For $c = 2$ and $n \geq 1$, $0 \leq 2^{n+1} \leq c \cdot 2^n$.

- $Is 2^{2n} = O(2^n)$?

No. Assume that there exist positive constants c and n_0 such that $0 \leq 2^{2n} \leq c \cdot 2^n$ for all $n \geq n_0$.

Then, $0 \leq 2^{(n+n)} \leq c \cdot 2^n$.

$0 \leq 2^n \cdot 2^n \leq c \cdot 2^n$

$0 \leq 2^n \leq c$. There do not exist positive constants c and n_0 such that $c \geq 2^n$ for all $n \geq n_0$.

• **Prob. 3-2 (a) and (b). Justify your answers mathematically according to the formal/limit definitions of big-oh, etc.**

- (a)

A	B	O	o	Ω	ω	Θ
$\lg^k n$	n^e	Yes	Yes $\lim_{n \rightarrow \infty} \frac{\lg^k n}{n^e} = 0$	No	No	No

- (b)

A	B	O	o	Ω	ω	Θ
n^k	c^n	Yes	Yes $\lim_{n \rightarrow \infty} \frac{n^k}{c^n} = 0$	No	No	No

• **Prob. 4-1 (a), (c), (e), and (g).**

- (a)

$f(n) = n^3 = \Omega(n^{\log_2 2 + 2})$ and $2(n/2)^3 \leq (1/4)n^3$ for all sufficiently large n . Thus, $T(n) = \Theta(n^3)$ by the Master theorem – case 3.

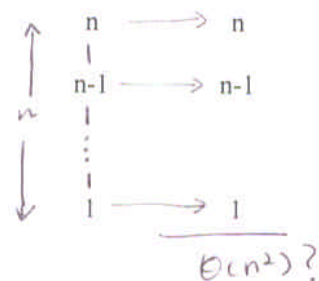
- (c)

$f(n) = n^2 = \Theta(n^{\log_4 16})$. Thus, $T(n) = \Theta(n^{\log_4 16} \lg n) = \Theta(n^2 \lg n)$ by the Master theorem – case 2.

- (e)

$f(n) = n^2 = O(n^{\log_2 7 - \log_2 \frac{7}{4}})$. Thus, $T(n) = \Theta(n^{\log_2 7})$ by the Master theorem – case 1.

- (g)



(i) $T(n) = T(n-1) + n$
 $\leq c(n-1)^2 + n = cn^2 + (1-2c)n + c$
 $\leq cn^2$ for $c=1$ and $n \geq 2$. Thus, $T(n) = O(n^2)$.

(ii) $T(n) = T(n-1) + n$
 $\geq c(n-1)^2 + n = cn^2 + (1-2c)n + c$
 $\geq cn^2$ when $c = 1/2$ and $n \geq 1$. Thus $T(n) = \Omega(n^2)$.

By (i) and (ii), $T(n) = \Theta(n^2)$.