Miray YILDIZ
161044023

CSE 321
Homework 3

1)

a) $T(n) = 27\,T(n/3) + n^2$

Apply a master theorem.

Master Theorem:

$T(n) = a\,T(n/b) + f(n) \longrightarrow f(n) = n^c$

Case 1: If $f(n) = \Theta(n^c)$ where $c < \log_b a$ then $T(n) = \Theta(n^{\log_b a})$

Case 2: If $f(n) = \Theta(n^c)$ where $c = \log_b a$ then $T(n) = \Theta(n^c \log n)$

Case 3: If $f(n) = \Theta(n^c)$ where $c > \log_b a$ then $T(n) = \Theta(f(n))$

so,

$a = 27$ $\quad \log_3 27 = \log_3 3^3 = 3$

$b = 3$ $\quad 3 > 2 \longrightarrow \Theta(n^{\log_3 27}) \rightarrow \underline{\Theta(n^3)}$

$f(n) = n^2$

b) $T(n) = 9\,T(n/4) + n$

Apply a master theorem

$a = 9$ $\quad \log_4 9 = \log_2 3$

$b = 4$ $\quad \longrightarrow \underline{\Theta(n^{\log_2 3})}$

$f(n) = n$ $\quad \log_2 3 > 1$

c) $T(n) = 2\,T(n/4) + \sqrt{n}$

Apply a master theorem.

$\sqrt{n} = n^{1/2}$

$a = 2$ $\quad \log_4 2 = 1/2$

$b = 4$ $\quad \frac{1}{2} = \frac{1}{2} \longrightarrow \Theta(n^{1/2} \log n) \rightarrow \underline{\Theta(\sqrt{n} \log n)}$

$f(n) = \sqrt{n}$

CamScanner ile tarandı

d) $T(n) = 2T(\sqrt{n}) + 1$

Assume $n = 2^k$

$T(2^k) = 2T(2^k)^{1/2} + 1$

$T(2^k) = 2T(2^{k/2}) + 1$

Assume $T(2^k) = S(k)$

$S(k) = 2S(k/2) + 1$ → Apply Master Theorem

$a = 2$
$b = 2$
$f(n) = 1$

$\log_2 2 = 1$
$1 > 0$

$\Theta(k)$ but $S(k) = T(2^k)$

$T(2^k) = \Theta(k)$ → $n = 2^k$
$k = \log n$

So, answer is $\Theta(\log n)$

e) $T(n) = 2T(n-2)$, $T(0) = 1$, $T(1) = 1$

$T(n) = 2T(n-2)$ → $T(n-2) = 2T(n-2-2)$ → $2T(n-4)$

$T(n) = 2.2T(n-4)$ → $T(n-4) = 2T(n-4-2)$ → $2T(n-6)$

$T(n) = 2.2.2T(n-6)$ → $T(n-6) = 2T(n-6-2)$ → $2T(n-8)$

$T(n) = 2.2.2.2.T(n-8)$ → $T(n-8) = 2T(n-8-2)$ → $2T(n-10)$

------ ------

$T(n) = 2^k . T(n-2k)$

$n = 2k$ → $T(n) = 2^{n/2} . T(2k-2k)$

→ $T(n) = 2^{n/2} . T(0)$ → $T(n) = 2^{n/2} . 1$

Answer is ⇒ $T(n) = \Theta(\sqrt{2}^n)$

f) $T(n) = 4T(n/2) + n$, $T(1) = 1$

Apply Master Theorem

$a = 4$
$b = 2$
$f(n) = n$

$\log_2 4 > 1$
$2 > 1$

$\Theta(n^{\log_2 4})$ → $\Theta(n^2)$

CamScanner ile tarandı

9) $T(n) = 2T(\sqrt[3]{n}) + 1$, $T(3) = 1$

Suppose $n = 3^m$ $m = \log_3 n$

$T(3^m) = 2T(3^{m/3}) + 1$

Suppose $S(m) = T(3^m)$

$S(m) = 2T(\sqrt[3]{3^m}) + 1$ $\qquad S(m) = 2T(3^{m/3}) + 1 \Rightarrow T(3^{m/3}) = S(m/3)$

$S(m) = 2S(m/3) + 1$

$a = 2$ $\qquad \log_3 2 < 1$ $\qquad f(m) = O(m^{\log_3 2}) \Rightarrow S(m) = \theta(m^{\log_3 2})$

$b = 3$

$f(m) = 1$

$n = 3^m$ $m = \log_3 n$

$S(m) = T(3^m) = \theta(m^{\log_3 2})$

$T(n) = \theta(\log_3 n)^{\log_3 2}$

Other questions on other pages

CamScanner ile tarandı

2)

I assume that n is a power of 2.

n	print_line("**") times
$n=1$ (2^0)	1 time
$n=2$ (2^1)	$1 * 1 = 1$ time $(2-1=1)$
$n=4$ (2^2)	$1 * 3 = 3$ times $(4-1=3)$
$n=8$ (2^3)	$3 * 7 = 21$ times $(8-1=7)$
$n=16$ (2^4)	$21 * 15 = 315$ times $(16-1=15)$
⋮	⋮
$n=n$	$T(n/2) * (n-1)$ times

$$T(n) = T(n/2) * (n-1) \longrightarrow T(n/2) = T(n/4) * (n/2 - 1)$$

$$T(n) = T(n/4) * (n/2 - 1) * (n-1) \longrightarrow T(n/4) = T(n/8) * (n/4 - 1)$$

$$T(n) = T(n/8) * (n/4 - 1) * (n/2 - 1) * (n-1)$$

⋮

$$T(n) = T(n/2^k) * \prod_{i=0}^{k-1} \left(\frac{n}{2^i} - 1\right) * (n-1)$$

Suppose $n = 2^k$ $\quad k = \log_2 n$

$$T(n) = \underbrace{T(n/n)}_{T(1)} * \prod_{i=0}^{(\log_2 n)-1} \left(\frac{n}{2^i} - 1\right)$$

$$T(n) = T(1) * \prod_{i=0}^{(\log_2 n)-1} \left(\frac{n}{2^i} - 1\right) \Rightarrow \prod_{i=0}^{\log_2 (\frac{n}{2})} \left(\frac{n}{2^i} - 1\right) \longrightarrow (n \cdot n \cdot n \cdots)$$

$T(1) = 1$

$$\underline{T(n) = \theta(n^{\log n})}$$

CamScanner ile tarandı

3)

Recurrence as follows

$T(n) = 3T\left(\frac{2}{3}n\right) + 1$

→ Recursively calls

→ Each time on array whose size is 2/3 of the original array size

Use Iteration method:

$T(n) = 3T\left(\frac{2}{3}n\right) + 1$

$T(n) = 9T\left(\frac{4n}{9}\right) + 1 + 3$

. . . .

$$T(n) = 1 + 3 + 3^2 + \dots + 3^{\log_{3/2} n}$$

$$= \frac{3^{\log_{3/2} n + 1} - 1}{3 - 1}$$

$$= \Theta\left(3^{\log_{3/2} n}\right)$$

$$= \Theta\left(3^{(\log_3 n)/(\log_3 3/2)}\right)$$

$$= \Theta\left(n^{1/\log_3 3/2}\right)$$

$$= \Theta\left(n^{2.71}\right)$$

CamScanner ile tarandı

4)

```
def insertionSort(A):
    length =len(A)
    count_is = 0
    for i in range (0,length):
        temp = A[i]
        J=i-1
        while J≥0 and temp<A[J]:
            A[J+1] = A[J]
            J=J-1
            count_is = count_is + 1        ##due to swap, increase count_is
        A[J+1]=temp
    return count_is


count_qs = 0
def partition(A, low, high):
    i = low - 1
    pivot = A[high]
    global count_qs
    for j in range(low, high):
        if A[j]<pivot:
            i=i+1
            A[i], A[j] = A[j], A[i]
            count_qs = count_qs + 1        ##due to swap, increase count_qs
    A[i+1], A[high] = A[high], A[i+1]      ## due to swap, increase count_qs
    count_qs = count_qs + 1
    return (i+1)

def quicksort(A, l, h):
    if l<h:
        index = partition(A, l, h)
        quicksort(A, l, index-1)
        quicksort(A, index+1, h)
```

CamScanner ile tarandı

In this question, I implement 2 sorting algorithms, Quick sort and Insertion sort I also kept the number of swaps in these algorithms.

Insertion sort performs two operations; it scans through the list, comparing each pair of elements, and swaps elements if they are out of order. Each operation contributes to the running time of the algorithm. If the input array is already in sorted order, insertion sort compares $O(n)$ elements and performs no swaps. Therefore, insertion sort's best case is $O(n)$. The worst case for insertion sort will occur when the input list is in decreasing order. To insert the last element, we need at most $n-1$ comparisons and at most $n-1$ swaps. To insert second to last element, we need at most $n-2$ comparisons and most $n-2$ swaps, and so on. The number of operations needed to perform insertion sort is therefore $2 \times (1+2+ \dots n-1)$. To calculate the recurrence relation for this algorithm:

$$* \sum_{q=1}^{p} q = \frac{p(p+1)}{2} \rightarrow \frac{2(n-1)(n-1+1)}{2} = n(n-1)$$

When use the master theorem to solve this recurrence, the algorithm's complexity is $O(n^2)$. When analyzing algorithms, average case is $\theta(n^2)$.

Quicksort is a divide-and-conquer algorithm. It works by selecting a pivot element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The sub-arrays are sorted recursively. Quicksort's worst case occurs when the partition process always picks greatest or smallest element as pivot. Follow recurrence for worst case:

$$* T(n) = T(0) + T(n-1) + \theta(n) \Rightarrow T(n) = T(n-1) + \theta(n)$$

The solution of recurrence is $\theta(n^2)$

Best case of quicksort occurs when the partition process always picks the middle element as pivot. Recurrence of best case:

$$* T(n) = 2T(n/2) + O(n)$$

The solution of recurrence is $\theta(n\log n)$ (Master theorem)

To do average case analysis, we need to consider all possible permutation of array and calculate time.

* Average case of quick sort is $\theta(n\log n)$

CamScanner ile tarandı

In conclusion, if we use large arrays, insertion sort's swap count is greater than quick sort's. When we run sorting algorithms with large arrays, we see results that support this theory.
Insertion sort is faster for small arrays.

5)

a)

Recurrence for algorithm $\begin{cases} T(n) = 5T(n/3) + n^2 \\ T(1) = 1 \end{cases}$

Solution with Master Theorem

$a = 5$ $\quad \log_3 5 < 2 \quad$ So, $O(n^2)$ ✓
$b = 3$
$f(n) = n^2$

b)

Recurrence for algorithm $\begin{cases} T(n) = 2T(n/2) + n^2 \\ T(1) = 1 \end{cases}$

Solution with master Theorem

$a = 2$ $\quad \log_2 2 < 2 \quad$ so, $O(n^2)$
$b = 2$ $\quad 1 < 2$
$f(n) = n^2$

c)

Recurrence for algorithm $\begin{cases} T(n) = T(n-1) + n \\ T(1) = 1 \end{cases}$

Solution with Backward Subtitution Method

Solution on the $\longrightarrow$ Other Page

CamScanner ile tarandı

$T(n) = T(n-1) + n$ → First equation

$T(n) = T(n-1) + n$
$T(n-1) = T(n-2) + n-1$
$T(n-2) = T(n-3) + n-2$

$T(n) = \left(T(n-2) + n-1\right) + n$

$T(n) = T(n-2) + (n-1) + n$ → Second equation

$T(n) = \left(T(n-3) + n-2\right) + (n-1) + n$

$T(n) = T(n-3) + (n-2) + (n-1) + n$ → Third equation

⋮

$T(n) = T(n-k) + (n-(k-1)) + (n-(k-2)) + (n-(k-3)) \dots + (n-1) + n$

Assume $n-k=0 \rightarrow n=k$

$T(n) = T(n-n) + (n-n+1) + (n-n+2) \dots + (n-1) + n$

$T(n) = T(0) + 1 + 2 + \dots + (n-1) + n$

$T(n) = 1 + \frac{n(n+1)}{2} \longrightarrow O(n^2)$ ✓

∝ The running times of all recurrence are the same so I will get the same result no matter which one I choose.

CamScanner ile tarandı