NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR

Cachar, Assam

B.Tech. IVth Sem

Subject Code: CS206

Subject Name: Algorithms

Submitted By:

Name	: Subhojit Ghimire
Sch. Id.	: 1912160
Branch	: CSE – B

1. What is the smallest value of n such that an algorithm whose running time is $100n^2$ runs faster than an algorithm whose running time is 2^n on the same machine?

➔ For an algorithm with running time $100n^2$ to run faster than an algorithm with running time 2^n,

calculated value of $100n^2$ < calculated value of 2^n, where, n = 1, 2, 3, …

Placing and checking each value of n,

for, n=1, $100\,(1)^2 = 100 > 2^1 = 2$

for, n=2, $100\,(2)^2 = 400 > 2^2 = 4$

for, n=3, $100\,(3)^2 = 900 > 2^3 = 8$

So on, for, n=14, $100\,(14)^2 = 19600 > 2^{14} = 16384$

for, n=15, $100\,(15)^2 = 22500 < 2^{15} = 32768$

Therefore, at the smallest value of n = 15, the algorithm with running time $100n^2$ starts to run faster than the algorithm with the running time 2^n.

2. Consider the searching problem:

Input: A sequence of n numbers A = (a_1, a_2, …, a_n) and a value v.

Output: An index i such that v = A[i] or the special value NIL of v does not appear in A.

Write pseudocode for liner search, which scans through the sequence, looking for v. Using a loop invariant, prove that your algorithm is correct. Make sure that your loop invariant fulfils the three necessary properties.

➔ Pseudocode for linear search:

```
linearSearch (A[n], v) {
for (i=1 i <= n; i = i+1)
        if (A[i] == v)
                return i
return NULL
}
```

Proof that the algorithm is correct using Loop Invariant's Properties:

i. Initialisation: Initially the number sequence A is empty, i.e., none of the elements equals v.
ii. Maintenance: For every iteration of i = 1 to n.
iii. Termination: If the iterated element A[i] equals v, the loop is terminated and the value of i is returned as the element position. If the iteration is completed and the number sequence still doesn't contain element equal to v, a NULL value is returned informing that the number is not present.

3. Consider sorting n numbers stored in an array A by first finding the smallest element of A and exchanging it with the element in A[1]. Then find the second smallest element of A, and exchange it with A[2]. Continue in this manner for the first n-1 elements of A. Write the pseudocode for this algorithm, which is known as the selection sort. What loop invariant does this algorithm maintain? Why does it need to run for only the first n-1 elements, rather than n elements? Give the best case and worst case running times of selection sort in Θ notation.

➔ Pseudocode for selection sort:

```
selectionSort ( A[n] ) {
        for (i=1; i <= n-1; i = i+1) {
                minIndex = i
                        for (j = i+1; j <= n; j = j+1) {
                                if (A[j] < A[minIndex] and j != minIndex) {
                                        minIndex = j
                                        break
                                }
                        }
                Swap (A[i], A[minIndex])
        }
}
```

Loop Invariant:

i. The array is sorted for first i elements in the outer loop.
ii. minIndex is always the minimum value in A [i to j] in the inner loop.

The algorithm needs to run for only the first n-1 elements, rather than n elements, because the last iteration will compare A[n] with other elements in A [1 to n-1].

Best Case Running Time of Selection Sort = Θ (n^2)

Worst Case Running Time of Selection Sort = Θ (n^2)

4. Use mathematical induction to show that when n is an exact power of 2, the solution of the recurrence

$$T(n) = \begin{cases} 2 & , if\ n = 2 \\ 2T\left(\frac{n}{2}\right) + n & , if\ n = 2^k\ for\ k > 1 \end{cases}$$

Is T(n) = n lg n.

➔ When n = 2, $T(2) = 2 = 2 \lg 2$, which is true.

When n= 2^k, $T(2^k) = 2^k \lg 2^k$, where, k > 1 (Let)

Then, the formula must hold true for k + 1,

i.e., $T(2^{k+1}) = 2^{k+1} \lg 2^{k+1}$ (To Prove)

Taking LHS, $T(2^{k+1}) = 2T \frac{2^{k+1}}{2} + 2^{k+1}$

$= 2T(2^k) + 2(2^k)$

$= 2(2^k \lg 2^k) + 2(2^k)$

$= 2(2^k)(\lg 2^k + 1)$

$= 2^{k+1}(\lg 2^k + \lg 2)$

$= 2^{k+1} \lg 2^{k+1}$, which is RHS

Hence, verified.

5. Show that for any real constants a and b, where b>0, $(n + a)^b = \Theta(n^b)$.

➔ We have, $n + a \le 2n$, when $|a| \le n$

And, $n + a \ge \frac{n}{2}$, when $|a| \le \frac{n}{2}$

These give, $0 \le \frac{n}{2} \le n + a \le 2n$, when $n \ge 2|a|$

As b > 0, raising all the term to the power of b,

$$(0)^b \le \left(\frac{n}{2}\right)^b \le (n + a)^b \le (2n)^b$$

$$= \quad 0 \le \frac{n^b}{2^b} \le (n + a)^b \le 2^b n^b$$

This gives, $(n + a)^b = \Theta(n^b)$ as there exist $c_1 = \frac{1}{2^b}$, $c_2 = 2^b$ and $n_o = 2|a|$.

Hence, verified.

6. Is 2^{n+1} = O (2n)? Is 2^{2n} = O (2^n)?

➔ For the first case,

$$2^{n+1} = 2\ (2^n)$$

i.e., for $n \geq 1$ and any $c \geq 2$,

$$0 \leq 2^{n+1} \leq c\ (2^n)$$

Hence, Yes, $2^{n+1} = O\ (2^n)$

For the second case,

$$2^{2n} = (2^n)\ (2^n)$$

i.e., for 2^{2n} to be O (2^n), we need constant c such that,

$$0 \leq (2^n)\ (2^n) \leq c\ (2^n)$$

Clearly, constant c must be $c \geq 2^n$, which is not possible for any arbitrarily large value of n.

Hence, No, $2^{2n} \neq 2\ (2^n)$.

7. Which is asymptotically larger: lg (lg * n) or lg * (lg n)?

➔ Let,	lg * n = x
Mathematically,	lg * n = min $\{i \geq 0 : \lg^{(i)} n \leq 1\}$
So,	lg (lg * n) = lg x
And,	lg * (lg n) = x -1
Asymptotically,	x -1 > lg x => lg * (lg n) > lg (lg * n)
Therefore,	lg * (lg n) is asymptotically greater than lg (lg * n).

8. Show that the solution of T (n) = T (n-1) + n is O (n^2).

➔ Let, $T(n) \leq cn^2$, $\forall\ n \geq n_0$, where, c and n_0 are positive constants

So, $T\ (n-1) \leq c\ (n-1)^2$

This gives,

$$\begin{aligned} T\ (n) &\leq c\ (n-1)^2 + n \\ &= cn^2 - 2cn + c + n \\ &= cn^2 - (n\ (2c - 1) - c) \\ &\leq cn^2 \qquad \text{, when } (n\ (2c - 1) - c) \geq 0 \\ &= O\ (n^2) \end{aligned}$$

Hence, verified.

9. Show that the solution of T (n) = T ([n/2]) + 1 is O (lg n).

➔ Let, $T(n) \le c \lg(n-2)$, $\forall$ $n \ge n_0$, where c and n_0 are positive constants.

So, $T\left(\frac{n}{2}\right) \le c \lg\left(\frac{n}{2} - 2\right)$

This gives, $T(n) \le c \lg\left[\frac{n}{2} - 2\right] + 1$

$< c \lg\left[\frac{n}{2} - 2 + 1\right] + 1$

$= c \lg\left[\frac{n-2}{2}\right] + 1$

$= c \lg(n-2) - c \lg 2 + 1$

$= c \lg(n-2) - (c-1)$

$\le c \lg(n-2)$, when $c \ge 1$

$= O(\lg n)$

Hence, verified.

10. Show that the solution of T (n) = 2 T ([n/2] + 17) + n is O (n lg n).

➔ Let, $T(n) \le c(n-34)\lg(n-34)$, $\forall$ $n \ge n_0$, where c and n_0 are positive constants.

So, $T\left(\frac{n}{2}\right) \le c\left(\left[\frac{n}{2}\right] - 34\right) lg\left(\left[\frac{n}{2}\right] - 34\right)$

This gives, $T(n) \le 2c\left(\left[\frac{n}{2}\right] + 17 - 34\right)\lg\left(\left[\frac{n}{2}\right] + 17 - 34\right) + n$

$= 2c\left(\left[\frac{n}{2}\right] - 34\right)\lg\left(\left[\frac{n}{2}\right] - 17\right) + n$

$= 2c\left(\frac{n-34}{2}\right)\lg\left(\frac{n-34}{2}\right) + n$

$= c(n-34)\lg(n-34) - c(n-34)\lg 2 + n$

$= c(n-34)\lg(n-34) - ((c-1)n - 34c)$

$\le c(n-34)\lg(n-34)$, when $((c-1)n - 34c) \ge 0$

$= O(n \lg n)$

Hence, verified.

11. Using the master method in Section 4.5, you can show that the solution to the recurrence T (n) = 4T (n/3) + n is T (n) = Θ ($n^{\log_3 4}$). Show that a substitution proof with the assumption, T (n) ≤ c $n^{\log_3 4}$ fails. Then show how to subtract off a lower-order term to make a substitution proof work.

➔ Let, T (n) ≤ $cn^{\log_3 4}$ - dn , ∀ n ≥ n_0, where c and n_0 are positive constants.

So, $T\left(\frac{n}{3}\right) \leq c\left(\frac{n}{3}\right)^{\log_3 4} - d\left(\frac{n}{3}\right)$

This gives,

$$
\begin{aligned}
T(n) &\leq 4\left[c\left(\frac{n}{3}\right)^{\log_3 4} - d\left(\frac{n}{3}\right)\right] + n \\
&= 4c\left(\frac{n^{\log_3 4}}{3^{\log_3 4}}\right) - 4\left(\frac{dn}{3}\right) + n \\
&= 4c\left(\frac{n^{\log_3 4}}{4}\right) - 4\left(\frac{dn}{3}\right) + n \\
&= cn^{\log_3 4} - dn - \frac{dn}{3} + n \\
&= cn^{\log_3 4} - dn - \left(\frac{d}{3} - 1\right)n \\
&\leq cn^{\log_3 4} - dn \qquad \text{, for } \left(\frac{d}{3} - 1\right)n \geq 0 \\
&= \Theta\left(n^{\log_3 4}\right)
\end{aligned}
$$

Hence, verified.

Also,

To show: when assumed T (n) ≤ $cn^{\log_3 4}$, the substitution proof fails.

Let, T (n) ≤ $cn^{\log_3 4}$, ∀ n ≥ n_0, where c and n_0 are positive constants.

So, $T\left(\frac{n}{3}\right) \leq c\left(\frac{n}{3}\right)^{\log_3 4}$

This gives,

$$
\begin{aligned}
T(n) &\leq 4c\left(\frac{n}{3}\right)^{\log_3 4} + n \\
&= 4c\left(\frac{n^{\log_3 4}}{3^{\log_3 4}}\right) + n \\
&= 4c\left(\frac{n^{\log_3 4}}{4}\right) + n \\
&= cn^{\log_3 4} + n
\end{aligned}
$$

This solution cannot be further substituted. This shows our assumption fails.

12. Use a recursion tree to determine a good asymptotic upper bound on the recurrence T(n) = T (n -1) + T (n/2) + n. Use the substitution method to verify your answer.

➔ Recursion Tree Method to Determine Good Asymptotic Upper Bound:

When the input size is one less than the previous step,

$$T(n) = \sum_{i=0}^{n} c\,(n-i) = \sum_{j=n}^{0} cj = \sum_{j=0}^{n} cj = c.\frac{n(n+1)}{2} = O(n^2)$$

When the input size is half the previous step,

$$T(n) = \sum_{i=0}^{n} \frac{cn}{2^i} = cn\sum_{i=0}^{n}\left(\frac{1}{2}\right)^i \le cn\sum_{i=0}^{\infty}\left(\frac{1}{2}\right)^i = cn.\frac{1}{1-\frac{1}{2}} = 2cn = O(n)$$

Clearly, $O(n^2)$ is a good asymptotic upper bound.

Verification:

Let, $T(n) \ge cn^2$

So, $T(n) \ge c(n-1)^2 + c\left(\frac{n}{2}\right)^2 + n$

$= cn^2 - 2cn + c + c\frac{n^2}{4} + n$

$= \frac{5}{4}cn^2 + (1-2c)n + c$

$= cn^2 + (1-2c)n + s$

$\ge cn^2$, when $c \le \frac{1}{2}$

$= O(n^2)$

Hence, verified.

13. Use the master method to give tight asymptotic bounds for the following recurrences.

a. T (n) = 2T (n/4) + 1
b. T (n) = 2T (n/4) + $\sqrt{n}$
c. T (n) = 2T (n/4) + n
d. T (n) = 2T (n/4) + n^2

➔ For the given recurrences, let a = 2 and b = 4.

This gives, $n^{\log_b a} = n^{\frac{1}{2}} = \sqrt{n}$

a. f (n) = O (1). Therefore, T (n) = Θ ($\sqrt{n}$)
b. f (n) = O ($\sqrt{n}$). Therefore, T (n) = Θ ($\sqrt{n}$ lg n)
c. f (n) = O (n). Therefore, T (n) = Θ (n)
d. f (n) = O (n^2). Therefore, T (n) = Θ (n^2)