GCD Algorithms (Euclid’s, Consecutive, Modified)

ANALYSIS

1. For $m > 6$ Input size $= n$

a] Euclid Algorithm

Basic Operation : Division

$D_{best}(n) = 1 \longrightarrow$ Efficiency class : Constant

$D_{worst}(n) = \log n \longrightarrow$ Efficiency class : Logarithmic

b] Consecutive Interger Checking Method

Basic Operation : Division

$D_{best}(n) = 1 \longrightarrow$ Efficiency class : Constant

$D_{worst}(n) \approx n \longrightarrow$ Almost Linear

c] Modified Euclid's Algorithm

Basic Operation : Subtraction.

$S_{best}(n) = 1 \longrightarrow$ Efficiency class : Constant

$S_{worst}(n) = n \longrightarrow$ Efficiency class : Linear.

Searching (a) Sequential (b) Binary Recursive

a)

ANALYSIS:

1. Input size : n [no. of elements in array A[0..n-1]]
2. Basic Operation : Comparison

The basic operation count is different for different I/P of same size.

3. Analogous Result:

- Best case - Key to be searched is at first Index

  $C_{best}(n) = 1$; $C_{best}(n) \in \Theta(1)$

  Efficiency class = Constant

- Worst case - Either key to be searched is at last Index or it is not found in the list.

  $C_{worst}(n) = \sum_{i=0}^{n-1} 1 = n-1-0+1 = n$

  $C_{worst}(n) \in \Theta(n)$; Efficiency class = Linear.

b)

ANALYSIS:

1. Input Size : n [number of elements]
2. Basic Operation : comparison

The basic operation count is different for different I/p of same size

3. Analogous Result:

- Best case - Key to be searched is at middle index

  $C_{best}(n) = 1$; $C_{best}(n) \in \Theta(1)$

  Efficiency class = constant

- Worst case - Key to be searched is either at last/first index or not found in the list. Loop is executed $(\log_2(n+1))$ times

  $C_{worst}(n) = \log_2 n + 1$; $C_{worst}(n) = \in \Theta(\log(n))$

  Efficiency class = Logarithmic

Sorting (a) Selection (b) Bubble (c) Insertion

ANALYSIS

1. Input size : n [no. of elements]
2. Basic Operation : Comparison

Basic operation count is same for different inputs of same size array.

3. $C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$

$$= \sum_{i=0}^{n-2} [n-1-(i+1)+1] = \sum_{i=0}^{n-2} n-1-i$$

$$= (n-1)\sum_{i=0}^{n-2} 1 - \sum_{i=0}^{n-2} i$$

$$= (n-1)[n-2-0+1] - \frac{(n-2)^2}{2}$$

$$= (n-1)(n-1) - \left(\frac{n^2+4-4n}{2}\right) = \frac{2n^2+2-4n-n^2-4+4n}{2}$$

$$= \frac{n^2-2}{2} \approx \frac{n^2}{2}$$

$\therefore C(n) \in \Theta(n^2)$

Efficiency class = Quadratic

a)

ANALYSIS:

1. Input Size : n [number of elements]
2. Basic operation : comparisons

The basic operation count is same of all arrays of size n

- Best Case : When array is already sorted.

No. of Swaps = 0 ; No. of comparisons = n - 1

$C_{best}(n) = \sum_{i=0}^{n-2} = n-2-0+1 = n-1 = n$; $C_{best}(n) \in \Theta(n)$

Efficiency class = Linear.

- Worst Case : When elements are in decreasing order.

C_{worst} (No. of Swaps = No. of Comparisons.

$C_{worst}(n) = (n-1) + (n-2) + (n-3) + \cdots + 2 + 1$

$$= \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1 = \sum_{i=0}^{n-2} [(n-2+0-i+1)] = \sum_{i=0}^{n-2} [n-i-1]$$

$$= [(n-1)(n-1+1)]/2 = [n(n-1)]/2$$

$C_{worst}(n) \in \Theta(n^2)$

Efficiency class = Quadratic

b)

ANALYSIS

1. Input Size : n
2. Basic Operation : Comparison
3. Basic Operation count is different for different I/P of same size

Best Case : Sorted Array in increasing order.

$C_{best}(n) = \sum_{i=1}^{n-1} 1 = n-1-1+1 = n-1 \in \Theta(n)$

Efficiency class = Linear

Worst case : Sorted Array in decreasing order.

$C_{worst}(n) = \sum_{i=1}^{n-1}\sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i-1+1 = \frac{(n-1)(n-1+1)}{2} = \frac{n(n-1)}{2} \in \Theta(n^2)$

Efficiency class = Quadratic

Average Case : Random order of elements.

$C_{avg}(n) = \sum_{i=1}^{n-1}\left(\frac{i+1}{2}\right) = \frac{1}{2}\left[\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1\right] = \frac{1}{2}\left[\frac{(n-1)(n-1+1)}{2} + (n-1+1-1)\right]$

$= \frac{1}{2}\left[\frac{n(n-1)}{2} + (n-1)\right] = \frac{1}{2}\left[\frac{n(n-1)+2(n-1)}{2}\right] = \frac{1}{2}\left[\frac{(n-1)(n+2)}{2}\right]$

$= \frac{1}{4}(n^2+n-2) \in \Theta(n^2)$

Efficiency class = Quadratic

c)

Brute Force String Matching

ANALYSIS:

1. Input Size: Text of length 'n' and Pattern of length 'm'
2. Basic Operation: Comparison

The Basic operation count is different for different inputs of pattern for same size.

$C_{best}(n/m) \in \Theta(m)$

Efficiency class : Linear

$$C_{worst}(n/m) = \sum_{i=0}^{n-m} \sum_{j=0}^{m-1} 1 = \sum_{i=0}^{n-m} m-1+1 = m\sum_{i=0}^{n-m} \cancel{m}.1$$

$$= (n-m-0+1)m = mn-m^2+1 = mn$$

$C_{worst}(n/m) \in \Theta(mn)$

$C_{avg}(n/m) \in \Theta(n)$

Efficiency class : linear.

Merge Sort

ANALYSIS

1. Input Size : n
2. Basic operation : comparison

Basic operation count is different for different inputs of the same size

- Best case : Array is already sorted in Ascending Order

$C_{best}(n) = 2C(n/2) + C_{merge}(n)$, for $n>1$, $C(1)=0$

$= 2C(n/2) + n/2$

By Smothness theorem $n = 2^k$

$C_{best}(2^k) = 2C(2^{k-1}) + 2^{k-1}$

$= 2^2 C(2^{k-2}) + 2 \times 2^{k-1}$

$= 2^3 C(2^{k-3}) + 3 \times 2^{k-1}$

$\vdots$

$C_{best}(2^i) = 2^i C(2^{k-i}) + i \times 2^{k-1}$ let $k-i=0$

$k=i$

$C_{best}(2^k) = 2^k C(2^{k-k}) + k \times 2^{k-1}$

$= 2^k C(1) + k \times 2^{k-1}$

$C_{best}(2^k) = k \cdot 2^{k-1} = (n\log_2 n)/2$

$C_{best}(n) \in \Theta(n\log_2 n)$

Efficiency class : $n\log_2 n$ class

- Worst case : Array with smaller elements coming from the alternating sub-array.

$C_{worst}(n) = 2C(n/2) + C_{merge}(n)$, for $n>1$, $C_{worst}(1)=0$

$= 2C(n/2) + n-1$

By smoothness theorem $n = 2^k$

$C_{worst}(2^k) = 2C(2^{k-1}) + 2^k - 1$

$= 2^2 C(2^{k-2}) + 2 \times 2^k - 2 - 1$

$= 2^3 C(2^{k-3}) + 3 \times 2^k - 2^2 - 2^1 - 1$

$\vdots$

$C_{worst}(2^i) = 2^i C(2^{k-i}) + i \times 2^i - [2^{i-1} + 2^{i-2} + \dots + 2^k]$

let $k-i=0 \Rightarrow k=i$

$C_{worst}(2^k) = 2^k C(1) + k \cdot 2^k - (2^k - 1)$

$= 2^k(k-1) + 1$

$= n(\log_2 n - 1) + 1 = n\log_2 n$

$C_{worst}(n) \in \Theta(n\log_2 n)$

Efficiency class = '$n\log_2 n$' class

Quick Sort

ANALYSIS

1. Input size : n
2. Basic Operation : comparison

Basic operation count is different for different I/P of same size

Best case: Pivot chosen divides the array into two halves

$C_{best}(n) = 2C(n/2) + n \quad , \quad n > 1, \; C(1) = 0 \quad , \text{let } n = 2^k$

$C_{best}(2^k) = 2C(2^{k-1}) + 2^k$

$= 2^2 C(2^{k-2}) + 2^{k} + 2^k$

$= 2^3 C(2^{k-3}) + 3.2^k$

$\vdots$

$C_{best}(2^k) = 2^i C(2^{k-i}) + i.2^k$

let $k - i = 0 \Rightarrow k = i$

$C_{best}(2^k) = 2^k C(1) + k2^k = k.2^k$

$C_{best}(n) = n \log n$

Efficiency class = log linear.

Worst case: Pivot chosen leads to one subarray to be empty.

$C_{worst}(n) = (n+1) + n + (n-1) + (n-2) + \dots + 3$

$= \frac{(n+1)(n+2)}{2} - 3 \in \Theta(n^2)$

Efficiency class = Quadratic.

Average case :

$C_{avg}(n) = \frac{1}{2} \sum_{s=0}^{n-1} C_{avg}(s) + C_{avg}(n-s-1) + (n+1) \quad n > 1, \; C(1) = 0$

$= \frac{1}{2} \sum_{i=0}^{n-1} C(s) + C(n-s-1) + (n+1)$

$C_{avg}(n) = 2n \ln n = 1.39 n \log_2 n$

$C_{avg}(n) \in \Theta(n \log n)$

Efficiency class = log linear.

DFS (Graph)

ANALYSIS:

Adjacency Matrix, $C(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1 = n \sum_{i=0}^{n-1} 1 = n(n-1-0+1)$

$$C(n) \in \Theta(n^2)$$

$$\therefore C(n) \in \Theta(|V^2|)$$

Efficiency class: Quadratic

BFS (Graph)

ANALYSIS

Input size: n (number of vertices)

Adjacency Matrix, $C(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1 = n \sum_{i=0}^{n-1} 1 = n(n-1-0+1)$

$$C(n) \in \Theta(n^2)$$

$$\therefore C(n) \in \Theta(|V^2|)$$

Efficiency class: Quadratic

Best case: Sparse graph

Worst case: complete graph

When adjacency matrix is used growth remains the same

DFS-based Topological Sort

ANALYSIS:

1. Input Size: n (no. of vertices)
2. Basic Operation Count: $c(n) = \sum_{i=1}^{n-1} \sum_{j=1}^{n-1} 1 = \sum_{i=1}^{n-1} n = n^2$

$\therefore c(n) \in \theta(n^2) = \theta(|V^2|)$

Efficiency class = Quadratic

Source Removal Topological Sort

ANALYSIS:

- Outer loop executes 'V' no. of times and Inner loop will execute 'E' no. of times
- Best Case Complexity: $c(n) \in \theta(V+E)$
- Worst Case Complexity: $c(n) \in \theta(|V^2|)$

Heap Sort (Bottom-up)

ANALYSIS:

Basic Operation: Comparison

It is different for different input of same size.

Warshall's Algorithm (Transitive Closure)

ANALYSIS

1. Input size: n (no. of vertices)
2. Basic Operation count, C(n)

$$\Rightarrow C(n) = \sum_{k=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n} 1 = \sum_{k=1}^{n}\sum_{i=1}^{n}(n-1+1) = \sum_{k=1}^{n}\sum_{i=1}^{n} n$$

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

$$= n^2(n) = n^3$$

$\therefore C(n) = \Theta(n^3)$

Efficiency class: Cubic

Floyd's Algorithm (All-pairs shortest paths)

ANALYSIS:

1. Input size: n (no. of vertices)
2. Basic Operation count, C(n),

$$C(n) = \sum_{k=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n} 1 = \sum_{k=1}^{n}\sum_{j=1}^{n}(n-1+1)$$

$$= n\sum_{k=1}^{n}\sum_{j=1}^{n} 1 = n\sum_{k=1}^{n}(n-1+1) = n^2\sum_{k=1}^{n} 1$$

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

$\therefore C(n) = \Theta(n^3)$

Efficiency class = Cubic

Knapsack (Bottom-up DP)

ANALYSIS

1. Input Size: nW
2. Basic Operation count, $C(n) = \sum_{i=1}^{n} \sum_{j=1}^{W} 1 = \sum_{i=1}^{n} (W-1+1) = W \sum_{i=1}^{n} 1$

$$C(n) = \Theta(nW)$$

Knapsack (Memory Function / Top-down)

Analysis:

- Using DP each row and column is computed in constant amout of time i.e, $\Theta(nW)$.
- Using MF only the time complexity is optimized to $\Theta(n+W)$ but space complexity remains the same.

Prim's Algorithm (MST)

Dijkstra's Algorithm