COMP 3711 Course Notes

Design and Analysis of Algorithms

LIN, Xuanyu

ALGORITHMS

COMP 3711 Design and Analysis of Algorithms

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

September 7, 2023

1 Sorting Problem

1.1 Selection Sort

Algorithm 1: Selection Sort

Input: An array $A[1...n]$ of elements
Output: Array $A[1...n]$ of elements in sorted order (asending)

```
for i ← 1 to n − 1 do
    for j ← i + 1 to n do
        if A[i] > A[j] then
            swap A[i] and A[j]
        end
    end
end
```

Running Time: $\frac{n(n-1)}{2}$
Best-Case = Worst-Case: $T(n) = \Theta(\frac{n(n-1)}{2}) = \Theta(n^2)$

1.2 Insertion Sort

Algorithm 2: Insertion Sort

Input: An array $A[1...n]$ of elements
Output: Array $A[1...n]$ of elements in sorted order (asending)

```
for i ← 2 to n do
    j ← i − 1 while j ≥ 1 and A[j] > A[j + 1] do
        swap A[j] and A[j + 1]
    end
    j ← j − 1
end
```

Running Time: Depends on the input array, ranges between $(n-1)$ and $\frac{n(n-1)}{2}$
Best-Case: $T(n) = n - 1 = \Theta(n)$ (Useless)
Worst-Case: $T(n) = \Theta(\frac{n(n-1)}{2}) = \Theta(n^2)$ (Commonly-Used)
Average-Case: $T(n) = \Theta(\sum_{i=2}^{n} \frac{i-1}{2}) = \Theta(\frac{n(n-1)}{4}) = \Theta(n^2)$ (Sometimes Used)

1.3 Wild-Guess Sort

Algorithm 3: Wild-Guess Sort

Input: An array $A[1...n]$ of elements
Output: Array $A[1...n]$ of elements in sorted order (asending)
$\pi \leftarrow [4, 7, 1, 3, 8, 11, 5, ...]$ Create random permutation Check if $A[\pi[i]] \leq A[\pi[i+1]]$ for all $i = 1, 2, ..., n-1$ If yes, output A according to π and terminate else $Insertion - Sort(A)$

Running Time: Depends on the random generation, could be faster than the insertion sort.

1.4 Worst-Case Analysis

The algorithm's worst case running time is $O(f(n)) \implies$ On all inputs of (large) size n, the running time of the algorithm is $\leq c \cdot f(n)$.

The algorithm's worst case running time is $\Omega(f(n)) \implies$ There exists at least one input of (large) size n for which the running time of the algorithm is $\geq c \cdot f(n)$.

Thus, Insertion sort runs in $\Theta(n^2)$ time.

Notice

Selection sort, insertion sort, and wild-guess sort all have worst-case running time $\Theta(n^2)$. How to distinguish between them?

- Closer examination of hidden constants
- Careful analysis of typical expected inputs
- Other factors such as cache efficiency, parallelization are important
- Empirical comparison

Stirling's Formula

Prove that $\log(n!) = \Theta(n \log n)$

First $\log(n!) = O(n \log n)$ since:

$$\log(n!) = \sum_{i=1}^{n} \log i \leq n \times \log n = O(n \log n)$$

Second $\log(n!) = \Omega(n \log n)$ since:

$$\log(n!) = \sum_{i=1}^{n} \log i \geq \sum_{i=n/2}^{n} \log i \geq n/2 \times \log n/2 = n/2(\log n - \log 2) = \Omega(n \log n)$$

2 Divide & Conquer

Main idea of D & C: Solve a problem of size n by breaking it into one or more smaller problems of size less than n. Solve the smaller problems recursively and combine their solutions, to solve the large problem.

Example: Binary Search

Input: A sorted array $A[1, ..., n]$, and an element x

Output: Return the position of x, if it is in A; otherwise output nil

Idea of the binary search: Set $q \leftarrow$ middle of the array. If $x = A[q]$, return q. If $x < A[q]$, search $A[1, ..., q-1]$, else search $A[q+1, ..., n]$.

Algorithm 4: Binary Search

Input: Array $A[1...n]$ of elements in sorted order

`BinarySearch(`$A[], p, r, x$`)` (p, r being the left & right iteration, x being the element being searched) **if** $p > r$ **then**

  return *nil*

end

$q \leftarrow [(p + r)/2]$

if $x = A[q]$ **then**

  return q

end

if $x < A[q]$ **then**

  `BinarySearch(`$A[], p, q-1, x$`)`

end

else

  `BinarySearch(`$A[], q+1, r, x$`)`

end