CSE310 Lecture 1 Notes

John J Li

July 5, 2021

The Sorting Problem - Formal definition

Sorting problem:

- Input: a sequence of n numbers $< a_1, a_2, ..., a_n >$.
- Output: a permetation (reordering) [of the input] $< a'_1, a'_2, ..., a'_n >$ such that $a'_1 \leq a'_2 \leq a'_n$

Insertion Sort: Analogy to storing cards...

There may be many ways to sort cards, but one intuitive procedure is to always insert a card into the right place.

- Works naturally when you draw a stack of cards one-by-one and put them into your hand.
- Similarily, if you sort all cards in your hand $\rightarrow$ *"in place"* sorting

Computationally represent the procedure

Pseudocode: Liberal user of English; Use of indentation for block structure; Omission of error handling (e.g. check if an array is empty).

```
// Pseudocode for insertion sort

INSERTION-SORT(A)
for j=2 to A.length // A.length=n
    key = A[j]
    // Insert A[j] into sorted sequence A[1...j-1]
```

```
i=j-1
while i>0 and A[i]>key
    A[i+1] = A[i]
    i=i-1
A[i+1]=key
```

Example:

3	9	6	1	2	9 should be inserted after 3 - no change
3	9	6	1	2	6 should be inserted between 3 and 9
3	6	9	1	2	1 should be inserted before 3
1	3	6	9	2	2 should be inserted between 1 and 3
1	2	3	6	9	sorted array

Analysis of the Insertion-sort algorithm

When we analyze an algorithm, we mainly focus on the performance (running-time). The running time of the algorithm depends on vaiour factors' success.

- depends on the nature of the input (already sorted vs. reverse sorted)
- depends on the size(n) (6 vs 10^6 elements) $\rightarrow$ parameterized running time.
  - $T(n) = \leftarrow$ function of the size (n).

Different kinds of running time analysis:

- Worst-case.
  - Max time on the input size n given an upperbound guerentee to the user.
- Average case.
  - expected time over all possible input
    - running time of every input and find the average. We need to know the probability of each input $\rightarrow$ Input $\times$ probability of that input $=$ weighted average.

    we don't know the exact probability, therefore, we make assumptions
    - all inputs are equally likely – uniform distibution.
- Best case
  - works on some inputs only, no guerentee to the user.

Next question: how to eliminate hardware dependency from running time.

- parametierize the running time based on the input size and then give the growth of the running time rather than giving absolute value. $\rightarrow$ asymptotic analysis (notation)

Asymptotic Notation:

- ignore machine dependent constants
- observe the growth of the running time.

One of the commonly used notation in the asymptotic noation is the θ notation.

θ notation:

- drop lower order terms and ignore constants
  - $T(n) = 3n^3 + 50n^2 + n + 600$
    - drop $50n^2 + n + 600$
    - ignore 3 in front of $3n^3$
  - $T(n) = \theta(n^3)$
- this means when $n \rightarrow \infty$ $\theta(n^3) > \theta(n^2)$ but for certain sizes of input, higher order term may perform better.

Analyzing the running time of the insertion sort. It had a for loop and a while loop.

- $T(n)$ = amount of time while loop runs for each iteration and the number of times while loop is invoked
  - $= \sum_{j=2}^{n} \theta(j)$
    - This is the sum of constant numbers: $2 + 3 + 4 + ... + n \rightarrow$ Arithemic series
    - $T(n) = \theta(n^2)$
- So in worst case: $T(n) = \theta(n^2)$
- Best case: while loop will be executed only one time (k) constant time
  - $T(n) = \sum_{j=2}^{n}(k) = \theta(n)$

Importance of parameterized time complexity

- gives an idea about the order of growth rather than giving an absolute value
- doesn't depend on the computer (hardware)

Activities

Prove: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Proof. $\square$

Prove: $\sum_{i=2}^{n} i = \theta(n^2)$

Proof. $\square$