2) Lecture 4

28 July 2024 13:05

We focus on algorithmic problem, where the aim is convert specified input instance to specified output instance.

Consider Sorting problem:

Input: a set of n numbers $(a_1, a_2, \ldots a_n)$

Output: a set of n numbers $(b_1, b_2, \ldots b_n)$ such that

- $b_1 <= b_2 <= \ldots b_n$ and
- $(b_1, b_2, \ldots b_n)$ is a permutation of $(a_1, a_2, \ldots a_n)$

There are infinite input instance possible.

The same sets of inputs can be given to any algorithm that can solve the problem.

No matter what sorting algorithm we use, the set of possible inputs always remains the same.

What about Case:

- A case is a specific input instance given to the algorithm
- A given case can be better or worse for a given algorithm.
- Some algorithms always perform the same no matter what the inputs are, but some algorithms might do worse on some inputs.
- Every algorithm may have some best case and some worst case.

Algorithm Case Example:

- Finding minimum of an unsorted list: Every element need to compare to find min.

  So best case, worst case are same.

- Finding a given element in a given list: Algo. linear search
  Best case – First element is given elmt
  worst case – last " " "

Determining efficiency of algorithm as function of input size:

- Running time is computed by counting the number of primitive operations used in the algorithm
  $\underline{\underline{f(n)}}$

- Two interesting cases: best case and worst case

- The best case is the input that minimizes the function, and the worst case is the input that maximizes the function.

- Best case and worst case are very concrete things w.r.t. a given algorithm.

Why do we need bounds:

Consider that running time: $\underbrace{n^2 + (\log n)! + e^{\frac{\log n}{n}}}$

- For interpretability of running time function, we are interested in few specific type of functions
  Like constant, logarithmic, linear, quadratic, cubic, exponential

- An upper bound is a function that sits on top of another function

- A lower bound is a function that sits on below of another function

- When we talk about Big Oh and Big Omega notation, we don't care if the function is always above/below the other function. Just that **after a certain point they always remain above/below other function.**

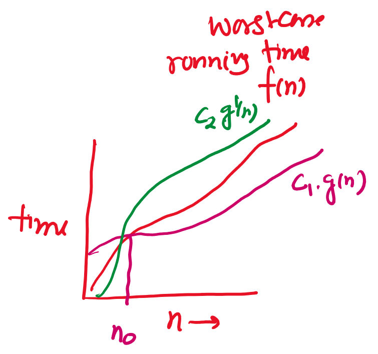
- There are infinite function possible: obvious upper bound it infinity and lower bound is zero.

- Least upper bound and greatest lower bound are the most informative.

$\underbrace{\text{worst/best case}}$ Input Instance on which Algo perform worst/best & we have captured running time of algo by a fn f(n)

$\underbrace{\text{upper/lower bound}}$ A function c. g(n) that bounds above/below f(n) after a certain point

Case is specific to algo.. Algo is fixed by now



Best/Worst Case + Lower/Upper Bound

Worst Case Lower Bound: A function that is a **boundary below the algorithms' runtime function**, when that algorithm is given the **input that maximize the running time**.

Worst Case Upper Bound: A function that is a **boundary above the algorithms' runtime function**, when that algorithm is given the **input that maximize the algorithm's run time**.

Best Case Lower Bound: A function that is a **boundary below the algorithms' runtime function**, when that algorithm is given the **inputs that minimize the algorithm's run time**.

Best Case Upper Bound: A function that is a **boundary above the algorithms' runtime function**, when that algorithm is given the **inputs that minimize the algorithm's run time**.

Examples:

Worst Case Lower Bound:

- **Comparison based sorting algorithm**: worst case lower bound is $\Omega(n \log n)$.

Best Case Lower Bound:

- **Insertion sort:** If the input is sorted, we only require to walk through the list without doing any insertions. Best case lower bound is $\Omega(n)$.

Worst Case Upper Bound:

- **Insertion sort:** If the input is sorted, in reverse order. Worst case upper bound is $O(n^2)$ ·

Best Case Upper Bound:

- **Finding an element in the list:** The best case is when the element we are looking for is the first element. The upper bound is $O(1)$.

While designing algorithms we mostly care about worst case lower/upper bound.

Sorting Problem

Alg 1. Insertion Sort

- best case : when input is already sorted (increasing order)
- worst case : " " " (decreasing order)

Alg :2 Quick Sort :

Sorting problem

- Insertion Sort
  - Best Case: Input is sorted
  - Worst Case: Input is sorted (in reverse order)
- Quick Sort
  - Worst Case: Input is sorted
  - Best Case: When input is sorted in reverse order