Introduction to Algorithms

What is an algorithm?

- A tool for solving a well-defined problem
- It takes input and produces output

How does one describe an algorithm?

(1) Define the problem. (What is the input, what is the output?)
(2) Describe the algorithm in words.
(3) Describe the algorithm in pseudo-code.
(4) Proof of correctness (Convince reader of correctness)
(5) Analysis (Runtime/space)

Example Problem: Sorting

1)
Input: Array $A[1 \ldots n]$ of numbers
Output: Sorted array A

(In other words, compute a permutation of the numbers in A such that they are sorted in increasing order)

2) Insertion Sort description:

(1) Maintain sorted portion at the beginning of the array.
(2) Iteratively insert the next element into the sorted portion of the array.

3) Insertion Sort pseudo-code:

Algorithm 1 int[] insertionSort(int $A[1 \ldots n]$)

```
j = 2;
while j <= n do
    key = A[j];
    i = j − 1;
    while (i > 0) and (A[i] > key) do
        A[i + 1] = A[i];
        i − −;
    end while
    A[i + 1] = key;
    j + +;
end while
return A;
```

Example of insertion sort:

5	2	4	6	1	3

How to prove correctness?

(1) Is it enough to give a concrete input, like the example $[5, 2, 4, 6, 1, 3]$ above, and trace it to prove the algorithm works?

(2) We can use this to find a counterexample which shows an algorithm does NOT work.
   (a) Think about simple examples
   (b) Think about examples with extremes of big and small
   (c) Think about examples with multiples of the same number
   (d) Failure to find a counterexample does NOT mean that the algorithm is correct

4) Proof of correctness using a loop invariant:

Algorithm 2 int[] insertionSort(int $A[1 \dots n]$)

```
j = 2;
while j <= n do
    //(I) A[1...j-1] are the elements originally in A[1...j-1] but sorted
    key = A[j];
    i = j - 1;
    while (i > 0) and (A[i] > key) do
        A[i + 1] = A[i];
        i - -;
    end while
    A[i + 1] = key;
    j + +;
end while
return A;
```

Above we have inserted a loop invariant (I).

In order to prove the algorithm is correct we first prove the loop invariant is true whenever we touch the top of the first while loop. We can then use the loop invariant to prove the overall algorithm is correct.

How do we prove that the loop invariant is true?

Induction (to prove that (I) is true for all $2 \leq j \leq n$)

Base Case: (Initialization)

Inductive Step: (Maintenance)

(I) is true before an iteration of the loop $\rightarrow$ (I) is true before the next iteration

Termination:

Loop terminates when $j =$ _____

Efficiency

(1) Correctness doesn't mean that the algorithm is a good one.
(2) Brute-force algorithms exist for most problems.
(3) For example: what if we sorted an array of numbers by randomizing the order repeatedly? When the numbers are sorted we stop.
   (a) This seems like this approach is obviously worse than insertion sort.
   (b) How do we show this?

5) Analysis of runtime — runtime dependent on n

Algorithm 3 int[] insertionSort(int $A[1 \ldots n]$)

```
j = 2;                                              1 time
while j <= n do                                     n time
    key = A[j];                                     (n-1) time
    i = j − 1;                                      (n-1) time
    while (i > 0) and (A[i] > key) do               Σ_{j=2}^{n} (tj + 1)
        A[i + 1] = A[i];                            Σ_{j=2}^{n} tj
        i − −;                                      Σ_{j=2}^{n} tj
    end while
    A[i + 1] = key;                                 (n-1) time
    j + +;                                          (n-1) time
end while
return A;                                           1 time
```

If we do above then next must happen reducing n by 1

t_j = # time we enter while loop for A + j

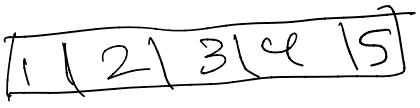
can be most

$\therefore$ $j = 2, 3, 4, 5, 6 \ldots, n, n+1$ → exit loop

$\therefore$ high − low + 1 (total numbers in range trick)

$\Rightarrow n + 1 - 2 + 1 = n$

$\therefore \sum_{j=2}^{n} (T_j + 1) = T_2 + T_3 + T_4 + \ldots + T_n$

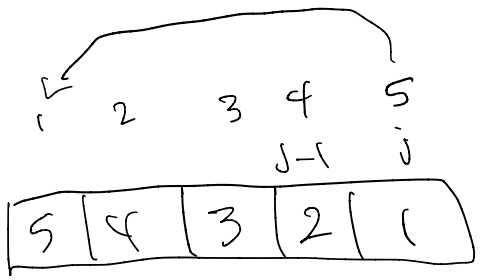


Best case: (Array already sorted)

$T_j = 0$ $\quad T_2 = T_3 = T_4 = T_5$

$$\sum_{j=2}^{n} (T_j + 1) = \sum_{j=2}^{n} (0+1) = \sum_{j=2}^{n} 1$$

$O(n-1) = O(n)$ $\quad = n - 2 + 1 = n - 1$



Worst case: (Array sorted in reverse order)

$T_j = j - 1$ # times we must go through loop is $j-1$

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

$$= \sum_{j=2}^{n} j = 2 + 3 + 4 + \ldots + n$$

n-1 #'s

$\Rightarrow O(n^2)$

Arithmetic series

$= (2 + n) + (3 + n-1) + (4 + n-2) \ldots$

# of copies of these numbers

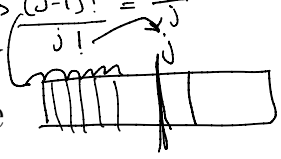
$= (n+2) + (n+2) + (n+2) \ldots$

$\Rightarrow \left(\frac{n-1}{2}\right) \cdot (n+2) \simeq n^2$

* Can solve in math heavy way here we do simpler method

What about the average case?

// elements in array have (j-1)! order the prior elements could be in. The jth element can be in j! order => $\frac{(j-1)!}{j!} = \frac{1}{j}$



There are $j-1$ elements in the sorted array. How many times on average will we have to go through the inner while loop to find the correct location of the current element?

times through inner loop

$\frac{1}{j}\,0 + \frac{1}{j}\,1 + \frac{1}{j}\,2 + \frac{1}{j}\,j-1 + \frac{1}{j}\,j-2 + \frac{1}{j}\,j-3$

// j possible values

$= \frac{j-1}{j} + \frac{j-1}{j} + \frac{j-1}{j}$

$= \frac{j}{2} \cdot \frac{j-1}{j} = \frac{j-1}{2}$

// j-1 is under 0, because that is max # of elements. which if we never enter inner loop stays the same (j-1)

// All occur at equal probability

Average case:

$\Rightarrow T_j = \frac{j-1}{2}$

$\Rightarrow \sum_{j=2}^{n} \left(\frac{j-1}{2} + 1\right) = \sum_{j=2}^{n} \frac{j-1}{2} + \underbrace{\sum_{j=2}^{n} 1}_{= (n-1)}$

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

$= \frac{1}{2}\left(\underbrace{\sum_{j=2}^{n} j}_{= \left(\frac{n-1}{2}\right)\cdot(n+2)} - \underbrace{\sum_{j=2}^{n} 1}_{= n-1}\right) + (n-1)$

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

↳ solve this as we skipped in lecture

$\in n^2$