Algorithm Design & Analysis III

Algorithm Analysis

- We are primarily concerned with time and space and particularly *how that grows* with respect to the input size.
- Although space usage can be a factor, time is usually how we classify algorithms, and particularly, we focus on the worst case.
- Other considerations: best case and average case.
- By runtime we mean
  - the amount of time it takes to run
  - the amount of work being done – cost
  - the number of operations being executed
  - (These should all be basically the same, at least in terms of growth rate.)

Example: Linear Search

Input: Array of size N, search element x

best-case: element is at index 0

$O(1)$

worst-case: element last (or not in array)

$\rightarrow O(N)$

Example: Binary Search

Input: sorted array of size N, search element x

best case: x is the (middle) median → O(1)

worst case: x is at the front, end, or not in the array

O(log N)

Best Case

- Provides a lower bound for the runtime.
- Determined by the easiest input, which makes it dependent on the make-up of the input (not just the size).
- Best Case runtimes should never be dependent on the size of the input! In other words, they should be true for all values of N (the input size).
- Notation: Sometimes uses big-Omega, sometimes big-Oh…but remember this is a tight bound.

Average Case

- Provides an estimate for the expected runtime given random input.
- So…not the “easiest” input and not the “worst” input.
- Does not provide a guarantee.
- Can be based on experimental analysis.
- Can be based on probability.
- Can help predict performance and can sometimes be more accurate in predictions than worst-case guarantees.
- Notation: Usually big-Oh

Worst Case

- Considers the absolute "hardest" worst possible input…
- …which provides a guarantee for the runtime.
- This is the most common because it is usually the most useful kind of analysis.
- Notation: Big-Oh, but remember this is a tight bound.

Cost Models

$$\binom{n}{3} = \frac{n!}{(n-3)!\,3!} = \frac{n(n-1)(n-2)}{6}$$

Actually counting the number of operations in an algorithm would be very tedious, so we usually just count up the # of times the innermost loop runs.

We have to choose wisely, though, as we need something that will give us an accurate count.

best: $O(n^3)$
worst: $O(n^3)$

```
Algorithm threeSum
Input: an array A of N integers
Output: ??? # of triples that add to 0

int count = 0
for(int i = 0; i < N; i++)
      for(int j = i + 1; j < N; j++)
            for(int k = j + 1; k < N; k++)
                  if(A[i] + A[j] + A[k] == 0)
                        count++
            end for
      end for
end for

return count
```

Example 1. Determine the final value of *sum* and then determine the best and worst-case runtimes of the pseudocode using appropriate notation.

independent loops

```
int sum = 0
for(int i = 0; i < N; i++)
    for(int j = 0; j < N; j++)
        sum++
    end for
end for
```

i: 0, 1, 2, ..., N-1

il: N + N + N + ... + N

best/worst: $O(N^2)$

$$\sum_{k=0}^{N-1} N = N^2$$

Example 2. Determine the final value of *sum* and then determine the best and worst-case runtimes of the pseudocode using appropriate notation.

```
int sum = 0
for i from 1 to N:
    for j from 1 to i:
        sum++
    end for
end for
```

Sum: $\frac{N(N+1)}{2}$

best: $O(N^2)$

worst: $O(N^2)$

i: 1, 2, 3 ... N

if: $1 + 2 + 3 + \ldots + N$

$$\sum_{k=1}^{N} k = \frac{N(N+1)}{2}$$

Example 3. Determine the final value of *sum* and then determine the best and worst-case runtimes of the pseudocode using appropriate notation. You can assume that N is a power of 2.

```
int sum = 0
for i from 1 to N:
    int j = 1
    while j <= N:
        sum++
        j = j * 2
    end while
end for
```

Sum: $N \log N + N$

best/worst: $O(N \log N)$

i: 1, 2, 3 ... N

if: $(\log N + 1) + (\log N + 1) + (\log N + 1) \ldots + (\log N + 1)$

$$\sum_{i=1}^{N} (\log N + 1) = \sum_{i=1}^{N} \log N + \sum_{i=1}^{N} 1$$

$$= N \log N + N$$

Example 4. Determine the final value of *sum* and then determine the best and worst-case runtimes of the pseudocode using appropriate notation. You can assume that N is a power of 4.

```
int sum = 0
while(N > 0):
    for i from 1 to N:
        sum++
    end for
    N = N/4
end while
```

sum : $\frac{4N-1}{3}$

best : $O(N)$

worst : $O(N)$

$N \quad \frac{N}{4} \quad \frac{N}{16} \quad \dots \quad 1$

$\log_4 N$

4^k

$N + \frac{N}{4} + \frac{N}{16} + \dots + 1$

$= 1 + 4 + 16 + \dots + \frac{N}{16} + \frac{N}{4} + N$

$= \sum_{k=0}^{\log_4 N} 4^k = \frac{4^{\log_4 N + 1} - 1}{3} = \frac{4N-1}{3}$

$O(N \log N)$

$O(N)$

$O(\log^2 N)$

Example 5. Determine the final value of *sum* and then determine the best and worst-case runtimes of the pseudocode using appropriate notation. You can assume that N is a power of 3.

```
int sum = 0
for(int i = 1; i <= N; i = i + 1)
    for(int j = 1; j <= i; j = j*3)
        sum++
    end for
end for
```

i : 1, 2, 3, ..., N

j : 1 + 1 + 2 + ... + $(\log_3 N + 1)$

$$\sum_{k=1}^{N} (\log_3 k + 1)$$

$$= \sum_{k=1}^{N} \log_3 k + \sum_{k=1}^{N} 1$$

$\log N!$ + N

best/worst: $O(N \log N)$

Example 6.

Determine the best and worst case runtimes for the pseudocode in terms of *N*.

```
int i = 1
while (i < N^2)
    int num = random(100)//a random number
from from 1 to 100
    if (num > 50)
        i = i * 3
    else
        i = i + 5
    end if
end while
```

$\log N^2 = 2 \log N$

best case : $O(\log N)$

worst case : $O(N^2)$

Example 7.

$(N-1) + (N-2) + \dots + 1 = \sum_{k=1}^{N-1} k = \frac{(N-1)(N)}{2}$

$O(N^2)$

Determine the best and worst case runtimes for the pseudocode in terms of *N*.

input size

```
A = an integer array of size N
for (int i = 0; i < N-1; i++)
    int j = i + 1
    while(j < N && A[i] < A[j])
        j++
    end while
    if(j < N)
        int temp = A[i]
        A[i] = A[j]
        A[j] = temp
    end if
end for
```

$O(1)$

best case: for the whole array

$A[j] \leq A[i]$

$O(N)$

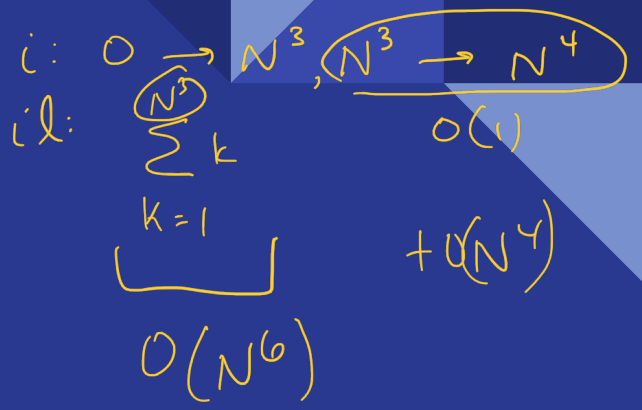
worst case: $O(N^2)$

$A[i] < A[j]$

Example 8.

Determine the best and worst case runtimes for the pseudocode in terms of *N*.

```
for (int i = 0; i < N^4; i++)
    for (int j = i; j < N^3; j++)
        //perform an θ(1) operation
    end for
end for
```



$O(N^6)$

$\frac{N^3(N^3+1)}{2}$ + N^4 N^3