Algorithm Design & Analysis

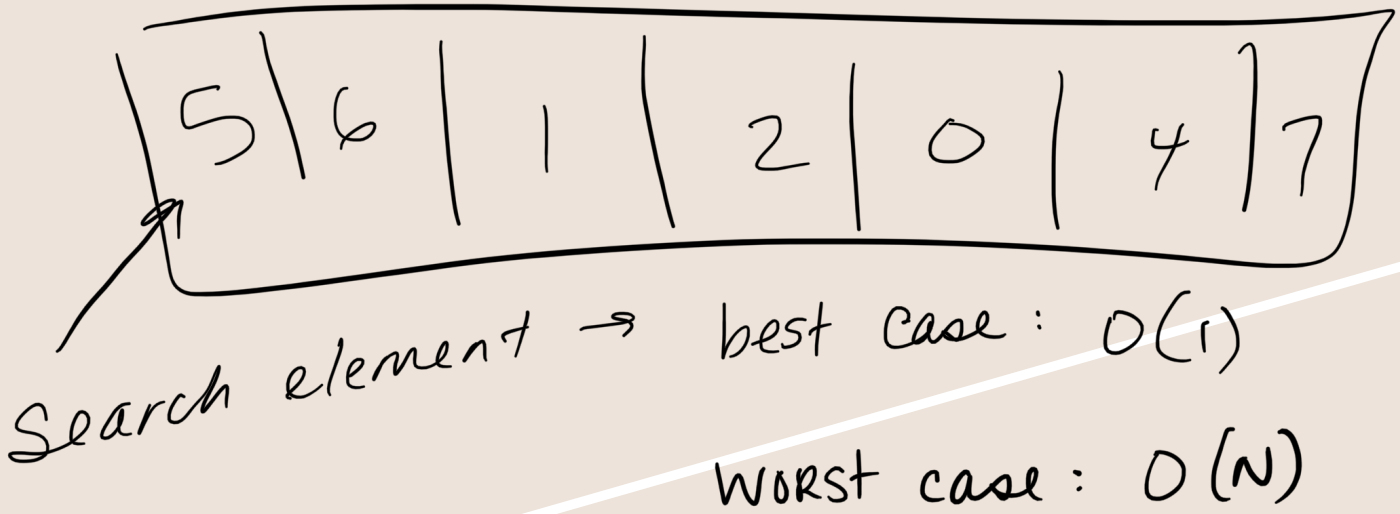
Part 2: Measuring Algorithm Efficiency

When we talk about algorithm efficiency, what are we concerned about?

Algorithm Analysis

- We are primarily concerned with space usage and time and particularly *how that grows* with respect to the input size.
- Although space usage can be a factor, runtime 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
  - the number of operations being executed
  - (These should all be basically the same, at least in terms of growth rate.)

Example: Linear Search



Example: Binary Search

best: search element is median

$O(1)$

WORST: $O(\log N)$

Best Case

- Provides a tight 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 mathematical analysis using probability.
- Can be based on experimental analysis.
- 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 worst possible input…
- …which provides a worst-case 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.

Reminders: P1 due Wed. 11:59 PM
Midterm 1: M 2/14

Cost Models

Actually counting the number of operations in an algorithm would be very tedious, so we usually just count up the number of times one operation occurs. We have to choose wisely, though, as we need something that will give us an accurate count.

```
Algorithm threeSum
Input: an array A of N integers
Output: ???

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.

best/worst : $O(N)$

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

$i: N, \frac{N}{4}, \frac{N}{16} + \dots + 1$

$i.1: \frac{N}{2}, \frac{N}{8}, \frac{N}{32} + \dots + 1$

```
int sum = 0
for (int i = N; i > 0; i = i / 4)
    for (int j = 0; j < i; j = j + 2)
        sum++
    end for
end for
```

$\frac{N}{2} + \frac{N}{8} + \frac{N}{32} + \dots + 16 + 4 + 1$

$= 1 + 4 + 16 + \dots + \frac{N}{32} + \frac{N}{8} + \frac{N}{2}$

$= \sum_{k=0}^{\log_4 N/2} 4^k = \frac{4^{\log_4 N/2 + 1} - 1}{3} = \frac{2N-1}{3}$

Example 2.

$N = 10k + 2$

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

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

```
int count = 0
for (int i = 1; i < N; i = i + 10)
    for (int j = 1; j < N; j = j * 2)
        count++
    end for
end for

while(count > 0)
    count = count/2
end while
```

$\log(N \log N)$

$\log M$

$i: 1, 11, 21, \ldots, N-1$

$\log N+1 \quad \log N+1 \quad \ldots \quad \log N+1$

$N-1$

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

$$= (\log N + 1) \sum_{k=1}^{\frac{N-1}{10}} 1$$

$$= (\log N + 1)\left(\frac{N-1}{10}\right)$$

$O(N \log N)$

Example 3.

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
```

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

$O(\log N) \rightarrow$ best

$O(N^2) \rightarrow$ worst

Example 4.

$N-1+N-2+N-3+\ldots+1 = \sum_{k=1}^{N-1} k = \frac{(N-1)(N)}{2}$

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

```
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
```

i=1 ... i=0

J=2 ... J=1

from 1 to N-1 ↓ N-1 + N-2

best case : $O(N)$

never true

worst case : $O(N^2)$

always true

Example 5.

$O(N^7)$ N^6

Determine the best and worst case runtimes for the pseudocode in terms of N. best/worst : $O(N^6)$

$i: 0 \to N^3$ $N^3 \to N^4$ $\to N^4 - N^3$

$\sim N^6 + N^4 - N^3$

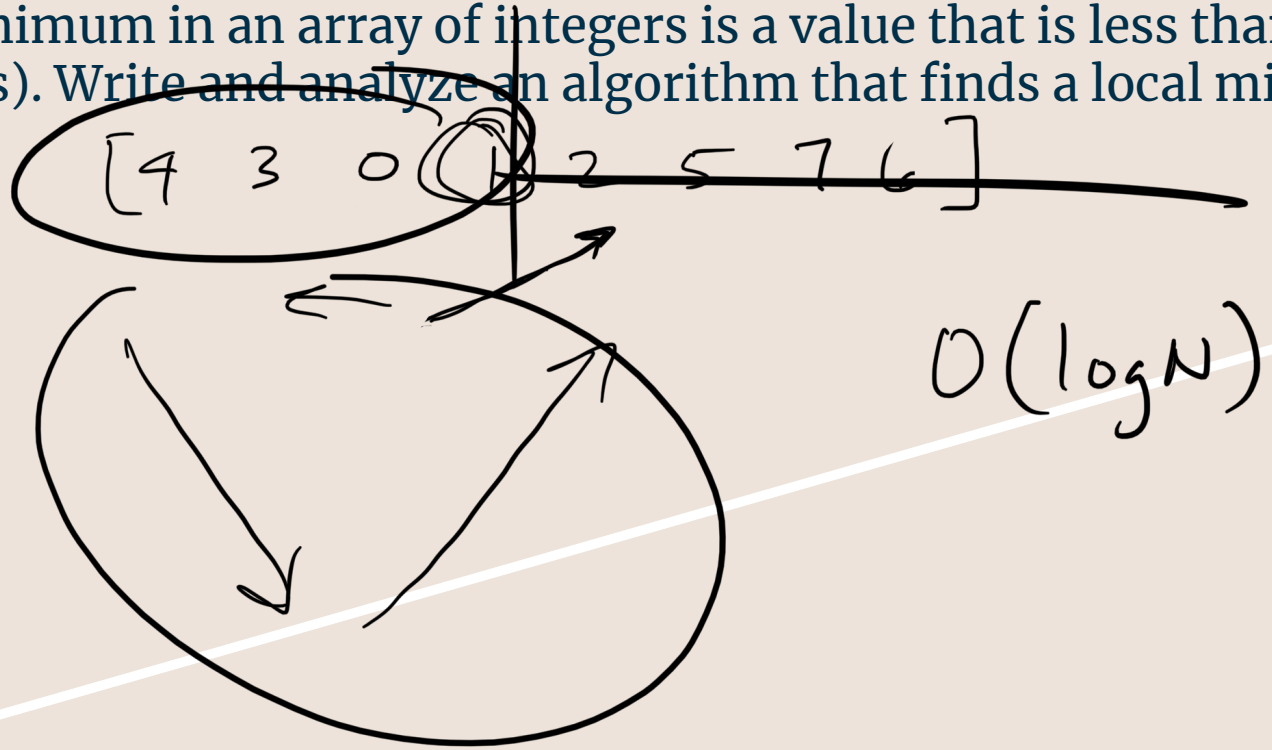
```
for (int i = 0; i < N^4; i++)
    for (int j = i; j < N^3; j++)
        //perform an θ(1) operation
    end for
end for
```

$\sim N^3 + N^3 - 1 + N^3 - 2 + \ldots + 1$

$= \sum_{k=1}^{N^3} k = \frac{N^3(N^3+1)}{2} \approx O(N^6)$

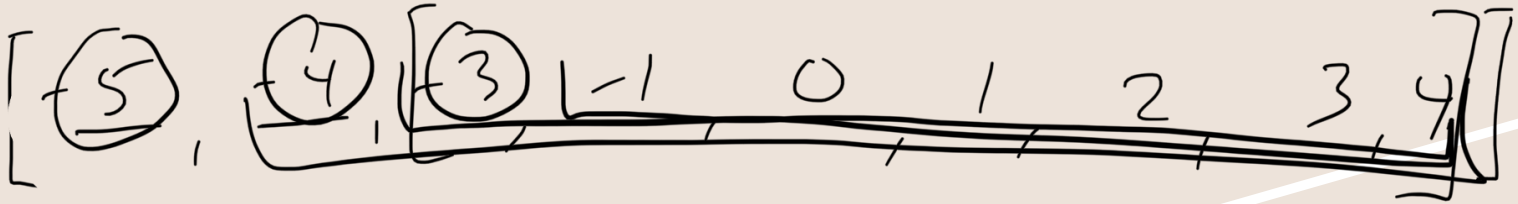
Example 6.

A local minimum in an array of integers is a value that is less than its neighbor(s). Write ~~and analyze~~ an algorithm that finds a local minimum in an array.



Example 7.

We saw a brute-force approach to the 3-sum problem. Write and analyze a new approach to the 3-sum problem. (Hint: Start by sorting the array. You don't have to say how, but sorting can be done in O(NlogN) time.)



binary search for each pair

$O(\log N)$ $O(N^2)$

$O(N^2 \log N)$