PA4 → due Thurs

Measuring Runtime

Count how many times each line executes, then say which Θ( ) statement(s) is(are) true.

```
int maxDifference(int[] arr){
  max = 0;                                  1
  for (int i=0; i<arr.length;  i++) {       n+1
    for (int j=0; j<arr.length; j++) {      n(n+1)
      if (arr[i] - arr[j] > max);           n*n
        max = arr[i] - arr[j];?             0 or n*n
    }
  }
  return max;
}
```

best case / worst case

Assume n = arr.length

A. f(n) = Θ(2^n)
B. f(n) = Θ(n^2)
C. f(n) = Θ(n)
D. f(n) = Θ(n^3)
E. Other/none/more

worst case

$4n^2 + 2n + 2n^3$

$4n^2 + 4n + 3$

$4n^2 + 4n^2 + 3 \rightarrow 8n^2 + 3$

$C = 8$, $N_0 = 3$ → $\Theta(n^2)$

Big O — ≤ — Upper bound
f(n) = O(g(n)), f(n) <= c * g(n)
for all n ≥ n0

Big Ω (omega) — ≥ — Lower bound
f(n) = Ω(g(n)), f(n) >= c * g(n)
for all n ≥ n0

Big Θ (theta) — tight bound
f(n) = Θ(g(n)), f(n) = c * g(n)
for all n ≥ n0

For each function in the list below, it is related to the function below it by O, and the reverse is not true. That is, n is O(n^2) but n^2 is not O(n).

- f(n) = 1/(n^2)
- f(n) = 1/n
- f(n) = 1
- f(n) = log(n)
- f(n) = sqrt(n)
- f(n) = n
- f(n) = n^2
- f(n) = n^3
- f(n) = n^4
- ... and so on for constant polynomials ...
- f(n) = 2^n
- f(n) = n!
- f(n) = n^n

Count how many times each line executes, then say which Θ( ) statement(s) is(are) true.

N = 100

```
int sumTheMiddle(int[] arr){
  int range = 100;                          range = 100
  int start = arr.length/2 - range/2;       start = N/2 - 50
  int sum = 0;                              1
  for (int i=start; i<start+range; i++)     100+1   100   start + range = N/2 - 50 + 100 = N/2 + 50
  {
    sum += arr[i];                          100
  }
  return max;
}
```

Assume n = arr.length

A. f(n) = Θ(2^n)
B. f(n) = Θ(n^2)
C. f(n) = Θ(n)
D. f(n) = Θ(1) constant time
E. None of these

N = 100
start = 100/2 - 50 = 0 → 100
start + range = 100/2 + 50 = 100

N = 10000
start = 10000/2 - 50 = 4950 → 100
start + range = 10000/2 + 50 = 5050

P(n) = 306
C = 306, g(n) = 1
N = 0

```
void printAllItemsTwice(int arr[], int size)
{
  for (int i = 0; i < size; i++) {          1  n+1  n   3n+2
    printf("%d\n", arr[i]); n
  }
  for (int i = 0; i < size; i++) {          1  n+1  n   3n+2
    printf("%d\n", arr[i]); n
  }
}
```

6n+4 → Θ(N)

What is the tight bound?

C = 6
N0 = 4

```
void printFirstItemThenFirstHalfThenSayHi100Times(int arr[], int size)
{
  printf("First element of array = %d\n",arr[0]);
  for (int i = 0; i < size/2; i++) {        n/2+1  n/2   3/2 n+3/2
    printf("%d\n", arr[i]); n/2
  }
  for (int i = 0; i < 100; i++) {           1  100+1  100   300+2
    printf("Hi\n"); 100
  }
}
```

3/2 n + 303/2 → Θ(N)

What is the tight bound?

```
void printAllNumbersThenAllPairSums(int arr[], int size)
{
  for (int i = 0; i < size; i++) {          1  n+1  n   3n+2
    printf("%d\n", arr[i]);n
  }
  for (int i = 0; i < size; i++) {          1  n+1  n
    for (int j = 0; j < size; j++) {        n+1  n(n+1)  n*n
      printf("%d\n", arr[i] + arr[j]); N
    }
  }
}
```

3n+2 + N(3n+2)

$3n^2 + 2n + 2n + 2$

$3n^2 + 4n + 2$

$\Theta(n^2)$

What is the tight bound?

Selection Sort

```
import java.util.Arrays;
public class Sort {
  public static void sortA(int[] arr) {
    for(int i = 0; i < arr.length; i += 1) {
      System.out.print(Arrays.toString(arr) + " -> ");
      int minIndex = i;
      for(int j = i; j < arr.length; j += 1) {
        if(arr[minIndex] > arr[j]) { minIndex = j; }
      }
      int temp = arr[i];
      arr[i] = arr[minIndex];
      arr[minIndex] = temp;
      System.out.println(Arrays.toString(arr));
    }
  }
}
```

N/2 · N

Selection Sort – what does it print out?

5

```
Sort.sortA(new int[]{ 53, 83, 15, 45, 49 });
```

```
[53, 83, 15, 45, 49] -> 15 83 53 45 49
                     -> 15 45 53 83 49
                     -> 15 45 49 83 53
                     -> 15 45 49 53 83
                     -> 15 45 49 53 83
```

4
3
2
1

Worst case: reverse sorted array
83, 53, 49, 45, 15

Best case: sorted array
15, 45, 49, 53, 83

What is the runtime? Consider the shape of the input array.

Worse case: Θ(N^2)
Best case: ~~Ω(N)~~ → Θ(N^2)

isSorted( )
Θ(N)

Insertion Sort

```
import java.util.Arrays;
public class Sort {
  public static void sortB(int[] arr) {
    for(int i = 0; i < arr.length; i += 1) {
      System.out.print(Arrays.toString(arr) + " -> ");
      for(int j = i; j > 0; j -= 1) {
        if(arr[j] < arr[j-1]) {
          int temp = arr[j-1];
          arr[j-1] = arr[j];
          arr[j] = temp;
        }
      }
      System.out.println(Arrays.toString(arr));
    }
  }
}
```

N · N/2

Insertion Sort – what does it print out?

```
Sort.sortB(new int[]{53, 83, 15, 45, 49 });
```

```
[53, 83, 15, 45, 49] -> 53 83 15 45 49
                     -> 53 83 15 45 49
                     -> 15 53 83 45 49
                     -> 15 45 53 83 49
                     -> 15 45 49 53 83
```

15 45 49 53 83 0
15 45 49 53 83 1
15 45 49 53 83 2
12 49 49 53 53 3
15 45 49 53 87 4
15 47 49 53 83

What is the runtime? Consider the shape of the input array.

Worse case: Θ(N^2)
Best case: Θ(N^2)