Motivation

```
1: function SeqSearch(A[],n,K)
2:     for i = 1 to n do
3:         if A[i] = K then
4:             return i
5:         end if
6:     end for
7:     return -1
8: end function
```

We will begin analyzing this algorithm by discussing the three notions of running time,

Best case The shortest running time for any input of size n. For `SeqSearch`, the input that results in the shortest running time is any array where the first element is K which takes $\Theta(1)$ time.

Worst case The longest running time for any input of size n. For `SeqSearch`, the input that results in the longest running time is any array that does not contain K which takes $\Theta(n)$ time.

Expected case The "average" running time assuming that the input is "random". To analyze the expected case running time we must make assumptions about the distribution of inputs so that we can quantify the probability of events. For this problem, let's assume that $K \in A$ and that all permutations of A are equally likely. That is, **Prob** $(K = A[i]) = \frac{1}{n}$ for $i = 1, 2, \ldots, n$. Having specified the probability of these events, we must also quantify the running time for each event,

Event	Time	Probability
$K = A[1]$	c	$\frac{1}{n}$
$K = A[2]$	$2c$	$\frac{1}{n}$
$K = A[3]$	$3c$	$\frac{1}{n}$
$K = A[4]$	$4c$	$\frac{1}{n}$
$\vdots$	$\vdots$	$\vdots$
$K = A[n-1]$	$c(n-1)$	$\frac{1}{n}$
$K = A[n]$	cn	$\frac{1}{n}$
$K \notin A$	$c(n+1)$	0

Let $ET(n)$ be the expected running time of `SeqSearch` with an input of size n, then we can compute the expected running time by evaluating the summation

$$ET(n) = \sum_{\text{Events}} \textbf{Prob}\,(\text{Event})\, t(\text{Event}) = \sum_{q=1}^{n} \textbf{Prob}\,(K = A[q])\, t(K = A[q]) = \sum_{q=1}^{n} \frac{1}{n}(cq)$$

This summation becomes a simple arithmetic sum,

$$ET(n) = \frac{c}{n}\sum_{q=1}^{n} q = \frac{c}{n} \cdot \frac{n(n+1)}{2} = \frac{c(n+1)}{2}.$$

Therefore, $ET(n) \in \Theta(n)$.

Introduction and Definitions

Definition 1. The expected value of a discrete random variable X with probability mass function p is

$$E(X) = \sum_{i=1}^{n} \textbf{Prob}\,(X = x_i)\, x_i \tag{1}$$

Definition 2. The expected run time of an algorithm is

$$ET(n) = \sum_{I} \textbf{Prob}\,(I)\, t(I) \tag{2}$$

where **Prob** (I) is probability of input/event I and $t(I)$ is the running time given input/event I.

Theorem 1. The expected value is a linear operator; suppose X_1 and X_2 are random variables and $a, b \in \mathbb{R}$, then

$$E(aX_1 + bX_2) = aE(X_1) + bE(X_2) \tag{3}$$

Theorem 2. The expected value of a random variable X can be computed using conditional expectation as follows:

$$E(X) = E(X \mid A)\textbf{Prob}\,(A) + E(X \mid \text{Not } A)(1 - \textbf{Prob}\,(A)) \tag{4}$$

Theorem 3. Suppose that X is a non-negative random variable (i.e. $X \subseteq \{0, 1, 2, \dots\}$), then

$$E(X) = \sum_{i=1}^{\infty} \textbf{Prob}\,(X \geq i) \tag{5}$$

Examples

Example 1. Determine the expected running time of the given algorithm. Assume that `Random`(n) generates a random integer between 1 and n with uniform distribution.

```
1: function Func1(A[],n)
2:     s = 0
3:     k = Random(n)
4:     for i = 1 to k do
5:         for j = 1 to k do
6:             s = s + A[i] · A[j]
7:         end for
8:     end for
9:     return s
10: end function
```

Best case The best case occurs when $k = 1$, then the algorithm takes $\Theta(1)$ time.

Worst case The worst case occurs when $k = n$, then the algorithm takes $\Theta(n^2)$ time.

Expected case Let $ET(n)$ be the expected running time of `Func1` with input of size n. Observe that $t(k = q) = cq^2$ and since `Random` samples uniformly **Prob** $(k = q) = \frac{1}{n}$, so

$$ET(n) = \sum_{q=1}^{n} \textbf{Prob}\,(k = q)\, t(k = q) = \sum_{q=1}^{n} \frac{1}{n}(cq^2).$$

This reduces to a sum of squares, for which there is a convenient closed-form expression,

$$ET(n) = \frac{c}{n}\sum_{q=1}^{n} q^2 = \frac{c}{n} \cdot \frac{n(n+1)(2n+1)}{6} = \frac{c(n+1)(2n+1)}{6}.$$

Therefore, $\boxed{ET(n) \in \Theta(n^2).}$

Example 2. Determine the expected running time of the given algorithm. Assume that `Random(`n`)` generates a random integer between 1 and n with uniform distribution.

```
1: function Func2(A[],n)
2:     s = 0
3:     for i = 1 to n do
4:         k = Random(i)
5:         for j = 1 to k^2 do
6:             s = s + A[i] · A[j]
7:         end for
8:     end for
9:     return s
10: end function
```

Best case The best case occurs when $k = 1$ every iteration and takes $\Theta(n)$ time.

Worst case The worst case occurs when $k = i$ every iteration and takes $\Theta(n^3)$ time.

Expected case Let $ET(n)$ be the expected running time of `Func2` with input of size n and let $T_{4-7}(i)$ be the running time of lines 4-7 of `Func2`. Observe that by linearity of expectation,

$$ET(n) = E\left[\sum_{i=1}^{n} T_{4-7}(i)\right] = \sum_{i=1}^{n} E\left[T_{4-7}(i)\right].$$

To compute $E\left[T_{4-7}(i)\right]$ we first observe that since the range of values for k is 1 to i, $\textbf{Prob}\left(k = q\right) = \frac{1}{i}$ and $t(k = q) = cq^2$, so

$$E\left[T_{4-7}(i)\right] = \sum_{q=1}^{i} \textbf{Prob}\left(k = q\right) t(k = q) = \sum_{q=1}^{i} \frac{1}{i}(cq^2)$$

So,

$$ET(n) = \sum_{i=1}^{n} E\left[T_{4-7}(i)\right] = \sum_{i=1}^{n}\sum_{q=1}^{i} \frac{1}{i}(cq^2) = \sum_{i=1}^{n} \frac{c}{i}\sum_{q=1}^{i} q^2.$$

We bound the summation above,

$$ET(n) = \sum_{i=1}^{n} \frac{c}{i}\sum_{q=1}^{i} q^2 \leq \sum_{i=1}^{n} \frac{c}{i}\sum_{q=1}^{i} i^2 = \sum_{i=1}^{n} ci^2 \leq \sum_{i=1}^{n} cn^2 = cn^3$$

and below

$$ET(n) = \sum_{i=1}^{n} \frac{c}{i}\sum_{q=1}^{i} q^2 \geq \sum_{i=1}^{n} \frac{c}{i}\sum_{q=\frac{i}{2}+1}^{i} \left(\frac{i}{2}\right)^2 = \sum_{i=1}^{n} \frac{ci^2}{8} \geq \frac{c}{8}\sum_{i=\frac{n}{2}+1}^{n} \left(\frac{n}{2}\right)^2 = \frac{cn^3}{64}.$$

Therefore, $\boxed{ET(n) \in \Theta(n^3).}$

Example 3. Determine the expected running time of the given algorithm. Assume that `Random(`n`)` generates a random integer between 1 and n with uniform distribution.

```
1: function Func3(A[],n)
2:     s = 0
3:     k = Random(n)
4:     if k = 1 then
5:         for i = n to n^2 do
6:             s = s + A[i]
7:         end for
8:     else
9:         for i = 1 to n ⌊log_2(n)⌋ do
10:            s = s + A[i]
11:        end for
12:    end if
13:    return s
14: end function
```

Best case The best case occurs when $k \neq 1$ and takes $\Theta(n \log_2(n))$ time.

Worst case The worst case occurs when $k = 1$ and takes $\Theta(n^2)$ time.

Expected case Let $ET(n)$ be the expected running time of `Func3` with an input of size n. Here, we leverage conditional probability to contend with the if-statement,

$$ET(n) = E(T(n) \mid k = 1)\textbf{Prob}\,(k = 1) + E(T(n) \mid k \neq 1)\textbf{Prob}\,(k \neq 1)$$

To compute the conditional expected values, we must examine the code and compute the running time assuming that the condition is met. That is, $E(T(n) \mid k = 1) = c(n^2 - n + 1)$ since when $k = 1$ the if-statement on line 4 is always satisfied and the for loop on lines 5-7 takes $c(n^2 - n + 1)$ time. So,

$$\begin{aligned} ET(n) &= E(T(n) \mid k = 1)\textbf{Prob}\,(k = 1) + E(T(n) \mid k \neq 1)\textbf{Prob}\,(k \neq 1) \\ &= c(n^2 - n + 1)\left(\frac{1}{n}\right) + cn\log_2(n)\left(1 - \frac{1}{n}\right) \\ &= cn - c + \frac{1}{n} + cn\log_2(n)\left(\frac{n-1}{n}\right) \\ &= cn - c + \frac{1}{n} + c(n-1)\log_2(n). \end{aligned}$$

Therefore, $\boxed{ET(n) \in \Theta(n\log(n)).}$

Example 4. Determine the expected running time of the given algorithm. Assume that `Random(`n`)` generates a random integer between 1 and n with uniform distribution.

```
1: function Func4(A[],n)
2:     s = 0
3:     k = Random(n)
4:     if k ≤ √n then
5:         for i = n to n^2 do
6:             s = s + A[i]
7:         end for
8:     else
9:         for i = 1 to n ⌊log_2(n)⌋ do
10:            s = s + A[i]
11:        end for
12:    end if
13:    return s
14: end function
```

Best case The best case running time occurs when $k > \sqrt{n}$ and takes $\Theta(n \log(n))$ time.

Worst case The worst case running time occurs when $k \leq \sqrt{n}$ and takes $\Theta(n^2)$ time.

Expected case Let $ET(n)$ be the expected running time of `Func4` with an input of size n. We again use conditional expectation to contend with the if-statement,

$$ET(n) = E(T(n) \mid k \leq \sqrt{n})\textbf{Prob}\left(k \leq \sqrt{n}\right) + E(T(n) \mid k > \sqrt{n})\textbf{Prob}\left(k > \sqrt{n}\right).$$

We shall approximate **Prob** $(k \leq \sqrt{n})$ by assuming that n is a perfect square so that **Prob** $(k \leq \sqrt{n}) = \frac{\sqrt{n}}{n}$,

$$\begin{aligned} ET(n) &= E(T(n) \mid k \leq \sqrt{n})\textbf{Prob}\left(k \leq \sqrt{n}\right) + E(T(n) \mid k > \sqrt{n})\textbf{Prob}\left(k > \sqrt{n}\right) \\ &= c(n^2 - n + 1)\left(\frac{\sqrt{n}}{n}\right) + cn\log_2(n)\left(1 - \frac{\sqrt{n}}{n}\right) \\ &= cn^{3/2} - c\sqrt{n} + \frac{c}{\sqrt{n}} + cn\log_2(n) - c\sqrt{n}\log_2(n). \end{aligned}$$

Therefore, $\boxed{ET(n) \in \Theta(n^{3/2}).}$

Example 5. Determine the expected running time of the given algorithm. Assume that `Random(`n`)` generates a random integer between 1 and n with uniform distribution.

```
1:  function Func5(A[],n)
2:      if n = 1 then
3:          return 0
4:      else
5:          s = 0
6:          for i = 1 to ⌊n/2⌋ do
7:              s = s + A[i]
8:          end for
9:          k = Random(n)
10:         if k is even then
11:             s = s + Func5(A[], n − 1)
12:         end if
13:         return s
14:     end if
15: end function
```

Best case The best case occurs when k is odd on the first call of `Func5`. In this case, no recursive calls are made and the algorithm takes $\Theta(n)$ time.

Worst case The worst case occurs when k is **always** even and a recursive call is always made until the base case is reached. The worst case running time therefore obeys the recurrence relation $T(n) = cn + T(n-1)$ with base case $T(1) = c$ and takes $\Theta(n^2)$ time.

Expected case Let $ET(n)$ be the expected running time of `Func5` with an input of size n. We can express the expected running time of `Func5` using conditional expectation,

$$ET(n) = E(T(n) \mid k \text{ is even})\textbf{Prob}\,(k \text{ is even}) + E(T(n) \mid k \text{ is odd})\textbf{Prob}\,(k \text{ is odd})$$

and can then leverage linearity to take advantage of the fact that the expected running time of the recursive call made on line 11 is $ET(n-1)$. For convenience we assume $\textbf{Prob}\,(k \text{ is even}) = \textbf{Prob}\,(k \text{ is odd}) = \frac{1}{2}$ noting that this assumption becomes increasingly valid as n grows arbitrarily large. So, by linearity we have

$$\begin{aligned} ET(n) &= E(T(n) \mid k \text{ is even})\textbf{Prob}\,(k \text{ is even}) + E(T(n) \mid k \text{ is odd})\textbf{Prob}\,(k \text{ is odd}) \\ ET(n) &= (cn + ET(n-1)) \cdot \frac{1}{2} + (cn) \cdot \frac{1}{2} \\ ET(n) &= cn + \frac{1}{2} ET(n-1). \end{aligned}$$

Observe that this expression makes some intuitive sense; the for loop which takes cn time is deterministic and always occurs and the recursive call occurs half of the time! We have now reduced this to a problem

of analyzing a recurrence relation, so we proceed with substitutions:

$$ET(n) = cn + \frac{1}{2}ET(n-1)$$
$$ET(n) = cn + \frac{1}{2}\left(c(n-1) + \frac{1}{2}ET(n-2)\right) = cn + \frac{c(n-1)}{2} + \frac{1}{2^2}ET(n-2)$$
$$ET(n) = cn + \frac{c(n-1)}{2} + \frac{1}{2^2}\left(c(n-2) + \frac{1}{2}ET(n-3)\right) = cn + \frac{c(n-1)}{2} + \frac{c(n-2)}{2^2} + \frac{1}{2^3}ET(n-3).$$

In general,

$$ET(n) = \sum_{i=0}^{k-1} \frac{c(n-i)}{2^i} + \frac{1}{2^k}ET(n-k)$$

We choose k such that $n - k = 1$ which means that $k = n - 1$, so

$$ET(n) = \sum_{i=0}^{n-2} \frac{c(n-i)}{2^i} + \frac{1}{2^{n-1}}ET(1) = \sum_{i=0}^{n-2} \frac{c(n-i)}{2^i} + \frac{c}{2^{n-1}}.$$

We have yet to analyze summations that have *both* arithmetic and geometric elements, but we have the skills to proceed. We shall, in a sense, combine both of our bounding techniques,

$$ET(n) = \sum_{i=0}^{n-2} \frac{c(n-i)}{2^i} + \frac{c}{2^{n-1}} \leq \sum_{i=0}^{n-2} \frac{cn}{2^i} + c \leq \sum_{i=0}^{\infty} \frac{cn}{2^i} + c = 2cn + c$$

So, $ET(n) \in O(n)$. Furthermore, the expected running time is **always** bounded below by the best case running time, therefore $ET(n) \in \Omega(n)$. Thus, $\boxed{ET(n) \in \Theta(n).}$

Example 6. Determine the expected running time of the given algorithm. Assume that `Random(`n`)` generates a random integer between 1 and n with uniform distribution.

```
1: function Func6(A[],n)
2:     if n ≤ 2 then
3:         return A[1]
4:     else
5:         k1 = Random(n)
6:         k2 = Random(n)
7:         if k1 < k2 then
8:             return A[n]
9:         else
10:            s = Func6(A[], n − 1) + Func6(A[], n − 2)
11:            return s
12:        end if
13:    end if
14: end function
```

Best case The best case occurs when $k_1 < k_2$ on the first call of `Func6` in which case the algorithm takes $\Theta(1)$ time.

Worst case The worst case occurs when $k_1 \geq k_2$ for **every** recursive call. In this case, the worst case running time obeys the recurrence relation $T(n) = c + T(n-1) + T(n-2)$, which as we have seen with the Fibonacci sequence takes $\Omega(\sqrt{2}^n)$ time.

Expected case Let $ET(n)$ be the expected running time of `Func6` with an input of size n. We can express the expected running time of `Func6` using conditional expectation,

$$ET(n) = E(T(n) \mid k_1 < k_2)\textbf{Prob}\left(k_1 < k_2\right) + E(T(n) \mid k_1 \geq k_2)\textbf{Prob}\left(k_1 \geq k_2\right).$$

We note that for sufficiently large values of n, $\textbf{Prob}\left(k_1 < k_2\right) \approx \textbf{Prob}\left(k_1 \geq k_2\right) = \frac{1}{2}$. Therefore,

$$\begin{aligned}
ET(n) &= E(T(n) \mid k_1 < k_2)\textbf{Prob}\left(k_1 < k_2\right) + E(T(n) \mid k_1 \geq k_2)\textbf{Prob}\left(k_1 \geq k_2\right) \\
ET(n) &= c \cdot \frac{1}{2} + (c + ET(n-1) + ET(n-2)) \cdot \frac{1}{2} \\
ET(n) &= c + \frac{1}{2}ET(n-1) + \frac{1}{2}ET(n-2)
\end{aligned}$$

In order to analyze this recurrence relation, we must begin bounding,

$$\begin{aligned}
ET(n) &= c + \frac{1}{2}ET(n-1) + \frac{1}{2}ET(n-2) \\
ET(n) &\leq c + \frac{1}{2}ET(n-1)\frac{1}{2}ET(n-1) \\
ET(n) &\leq c + ET(n-1)
\end{aligned}$$

We now proceed with substitutions,

$$\begin{aligned}
ET(n) &\leq c + ET(n-1) \\
ET(n) &\leq c + c + ET(n-2) \\
ET(n) &\leq c + c + c + ET(n-3) \\
ET(n) &\leq kc + ET(n-k)
\end{aligned}$$

We choose k such that $n - k = 2$, so $k = n - 2$. So,

$$ET(n) \leq c(n-2) + ET(2) = c(n-2) + c.$$

Thus, $ET(n) \in O(n)$. We bound below similarly,

$$\begin{aligned}
ET(n) &= c + \frac{1}{2}ET(n-1) + \frac{1}{2}ET(n-2) \\
ET(n) &\geq c + \frac{1}{2}ET(n-2)\frac{1}{2}ET(n-2) \\
ET(n) &\geq c + ET(n-2)
\end{aligned}$$

We now proceed with substitutions,

$$\begin{aligned}
ET(n) &\geq c + ET(n-2) \\
ET(n) &\geq c + c + ET(n-4) \\
ET(n) &\geq c + c + c + ET(n-6) \\
ET(n) &\geq kc + ET(n-2k)
\end{aligned}$$

We choose k such that $n - 2k = 2$, so $k = \frac{n-2}{2}$. So,

$$ET(n) \geq \frac{c(n-2)}{2} + ET(2) = \frac{c(n-2)}{2} + c.$$

Thus, $ET(n) \in \Omega(n)$. Therefore, $\boxed{ET(n) \in \Theta(n).}$

```
1: function SortedInsert(A[],n,x)
2:     A[n+1] = x
3:     j = n
4:     while (j > 0) and (A[j] > A[j+1]) do
5:         Swap(A[j],A[j+1])
6:         j = j - 1
7:     end while
8: end function
```

Best case The best case occurs when $x \geq A[n]$ and `SortedInsert` takes $\Theta(1)$ time.

Worst case The worst case occurs when $x < A[1]$ and `SortedInsert` takes $\Theta(n)$ time.

Expected case In order to analyze the expected case running time, we must make assumptions about the distribution of the inputs. Assume

1. A has no duplicates.
2. $x \notin A$.
3. All locations of x in A are equally likely.

Let $ET_{SI}(n)$ be the expected running time of `SortedInsert` under these assumptions. We can now characterize the probability of x being inserted at a particular location,

Event	Time	Probability
$A[n] \leq x$	c	$\frac{1}{n+1}$
$A[n-1] \leq x < A[n]$	$2c$	$\frac{1}{n+1}$
$A[n-2] \leq x < A[n-1]$	$3c$	$\frac{1}{n+1}$
$A[n-3] \leq x < A[n-2]$	$4c$	$\frac{1}{n+1}$
$\vdots$	$\vdots$	$\vdots$
$A[2] \leq x < A[3]$	$c(n-1)$	$\frac{1}{n+1}$
$A[1] \leq x < A[2]$	cn	$\frac{1}{n+1}$
$x < A[1]$	$c(n+1)$	$\frac{1}{n+1}$

Having compute these probabilities and times, the expected running time is

$$ET_{SI}(n) = \frac{1}{n+1}(c + 2c + 3c + \cdots + cn + c(n+1)) = \frac{1}{n+1} \cdot \frac{c(n+1)(n+2)}{2} = \frac{c(n+2)}{2}$$

Therefore, $\boxed{ET_{SI}(n) \in \Theta(n).}$

1: **function** `InsertionSort`($A[]$,n)
2: **for** $i = 1$ to $n - 1$ **do**
▷ insert $A[i + 1]$ in $A[1..i]$
▷ maintains: $A[1] \leq A[2] \leq A[3] \leq \cdots \leq A[i]$
3: $x = A[i + 1]$
4: `SortedInsert`($A[]$,i,x)
5: **end for**
6: **end function**

Note that the running time of `InsertionSort` can be computed by evaluating the sum,

$$T(n) = \sum_{i=1}^{n} T_{SI}(i)$$

where $T_{SI}(i)$ is the running time of sorted insert on the array of size i.

Best case The best case occurs when A is already sorted and no swaps must occur in `SortedInsert`; the best case running time of `SortedInsert` is $\Theta(1)$, thus the best case running time of `InsertionSort` is $\Theta(n)$.

Worst case The worst case occurs when A is in reverse sorted order and each call to sorted insert takes $\Theta(i)$ time. Note that this is $\Theta(i)$ **NOT** $\Theta(n)$ since the size of the input to `SortedInsert` is i. Therefore, the worst case running time is $\sum_{i=1}^{n} ci = \frac{cn(n+1)}{2}$ which is in $\Theta(n^2)$.

Expected case Let $ET(n)$ be the expected case running time of `InsertionSort` on an array of size n. We assume that all permutations of A are equally likely – this ensures that the assumptions of our analysis of `SortedInsert` are satisfied. The expected running time is

$$ET(n) = E\left[\sum_{i=1}^{n} T_{SI}(i)\right]$$

and by linearity we have

$$ET(n) = \sum_{i=1}^{n} ET_{SI}(i) = \sum_{i=1}^{n} ci = \frac{cn(n+1)}{2}.$$

Note that this analysis seems to imply that the expected case and the worst case are identical, however the constants for the worst case and expected case are different. In particular, the constant for the expected case is approximately half that of the constant for the worst case since $ET_{SI}(i) = \frac{c(i+2)}{2}$.
Therefore, $\boxed{ET(n) \in \Theta(n^2).}$

```
1: function InsertionSortRec(A[],n)
2:     if n > 1 then
3:         InsertionSortRec(A[],n − 1)
           ▷ insert A[n] in A[1..(n − 1)]
4:         x = A[n]
5:         SortedInsert(A[],n − 1,x)
6:     end if
7: end function
```