Probabilistic Analysis I - Probabilistic Analysis for Sorting and Selection

Sequential Search

Input : Array A of n distinct integers.

Key K.

Output : p such that $A[p] = K$ or -1 if there is no such p.

```
function SeqSearch(A[ ],n,K)
for i ← 1 to n do
    if (A[i] = K) then
        return (i);
    end
end
/* Element x not found.                    */
return (−1);
```

Sequential Search: Expected Running Time

Expected running time =

$$\left(\sum_{q=1}^{n} \text{Prob}(\mathsf{A}[q] = K)\, t(\mathsf{A}[q] = K)\right) + \text{Prob}(K \notin \mathsf{A})\, t(K \notin \mathsf{A}).$$

Prob(I) = probability that event I occurs.
$t(I)$ = running time given that event I occurs.

Sequential Search: Expected Running Time

Assume $\mathsf{K} = \mathsf{A}[q]$ for exactly one q and all permutations are equally likely.

Event	Time	Probability
$\mathsf{K} = \mathsf{A}[1]$		
$\mathsf{K} = \mathsf{A}[2]$		
$\mathsf{K} = \mathsf{A}[3]$		
$\mathsf{K} = \mathsf{A}[4]$		
$\vdots$		
$\mathsf{K} = \mathsf{A}[n-1]$		
$\mathsf{K} = \mathsf{A}[n]$		
$\mathsf{K} \notin \mathsf{A}$		

Expected Running Time

Expected/average running time $ET(n) = \sum_I \text{Prob}(I)t(I)$;

- $\text{Prob}(I)$ = probability of event I;
- $t(I)$ = running time given event I.

$\text{Prob}(I)$ depends on the input probability distribution (usually uniform).

Expected Running Time

- $T_{\text{worst}}(n)$ = worst case running time on n inputs;
- $T_{\text{best}}(n)$ = best case running time on n inputs;
- $ET(n)$ = expected running time on n inputs.
  $ET(n)$ depends upon the input probability distribution (usually uniform.)

$$T_{\text{best}}(n) \leq ET(n) \leq T_{\text{worst}}(n).$$

Expected Running Time

Expected/average running time $ET(n) = \sum_I \text{Prob}(I)t(I)$;

- $\text{Prob}(I)$ = probability of event I;
- $t(I)$ = running time given event I.

$\text{Prob}(I)$ depends on the input probability distribution (usually uniform).

Example

```
Func1(A, n)
/* A is an array of integers                    */
s ← 0;
k ← Random(n);
for i ← 1 to k do
    for j ← 1 to k do
        s ← s + A[i] * A[j];
    end
end
return (s);
```

Random(n) generates a random integer between 1 and n with uniform distribution (every integer between 1 and n is equally likely.)

Example

```
Func2(A, n)
/* A is an array of integers                    */
s ← 0;
k ← Random(⌊log₂(n)⌋);
for i ← 1 to 2^k do
  | s ← s + A[i];
end
return (s);
```

Random(n) generates a random integer between 1 and n with uniform distribution (every integer between 1 and n is equally likely.)

Expectation

X is a random variable.

The expectation of X is:

$$E(X) = \sum_{I} \text{Prob}(X = I)\ I.$$

- Linearity of expectation:

$$E(X_1 + X_2) = E(X_1) + E(X_2).$$

- Conditional expectation:

$$E(X) = E(X \mid A)\ \text{Prob}(A) + E(X \mid \text{Not } A)\ (1 - \text{Prob}(A)).$$

- Formula for non-negative random variables ($X \in \{0, 1, 2, 3, \ldots\}$):

$$E(X) = \sum_{i=1}^{\infty} \text{Prob}(X \geq i).$$

Linearity of Expectation

X is a random variable.

The expectation of X is:

$$E(X) = \sum_{I} \text{Prob}(X = I)\ I.$$

Linearity of expectation:

$$E(X_1 + X_2) = E(X_1) + E(X_2).$$

$$E(\sum_{i=1}^{k} X_i) = \sum_{i=1}^{k} E(X_i).$$

Linearity of Expectation

function `Func3(A[ ],n)`

/* A *is an array of* n *integers* */

$s \leftarrow 0$;

for $i \leftarrow 1$ **to** n **do**

  $k \leftarrow$ `Random`(i);

  for $j = 1$ **to** k^2 **do**

    $s \leftarrow s + j \times A[\lceil j/n \rceil]$;

  end

end

return (s);

`Random`(i) generates a random integer between 1 and i with uniform distribution (every integer between 1 and i is equally likely.)

Conditional Expectation

X is a random variable.

The expectation of X is:

$$E(X) = \sum_I \text{Prob}(X = I)\, I.$$

Conditional expectation:

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

Conditional Expectation

function `Func4`(A[ ],n)
/* A *is an array of* n *integers* */
$k \leftarrow$ `Random`(n);
$s \leftarrow 0$;
if $(k = 1)$ **then**
| **for** $i \leftarrow n$ **to** n^2 **do** $s \leftarrow s + i \times A[\lceil i/n \rceil]$;
else
| **for** $i \leftarrow 1$ **to** $n \lfloor \log_2(n) \rfloor$ **do** $s \leftarrow s + i \times A[\lceil i/n \rceil]$;
end
return (s);

`Random`(n) generates a random integer between 1 and n with uniform distribution (every integer between 1 and n is equally likely.)

Conditional Expectation

function `Func5`(A[ ], n)

/* A *is an array of* n *integers* */

$k \leftarrow$ `Random`(n);

$s \leftarrow 0$;

if $(k \leq \sqrt{n})$ **then**

  for $i \leftarrow n$ **to** n^2 **do** $s \leftarrow s + i \times A[\lceil i/n \rceil]$;

else

  for $i \leftarrow n$ **to** $n\lfloor \log_2(n) \rfloor$ **do** $s \leftarrow s + i \times A[\lceil i/n \rceil]$;

end

return (s);

`Random`(n) generates a random integer between 1 and n with uniform distribution (every integer between 1 and n is equally likely.)

Example

```
Func6(A, n)
/* A is an array of integers                                  */
if (n = 1) then  return(0) ;
else
|   s ← 0;
|   for i ← 1 to ⌊n/2⌋ do
|   |   s ← s + A[i] ∗ A[n − i + 1];
|   end
|   k ← Random(n);
|   if (k is even) then
|   |   s ← s + Func6(A, n − 1);
|   end
|   return (s);
end
```

Example

```
Func7(A, n)
/* A is an array of integers                                */
if (n ≤ 2) then return(A[1]) ;
else
    k1 ← Random(n);
    k2 ← Random(n);
    if (k1 < k2) then
        return (A[n]);
    else
        s ← Func7 (A, n − 1) + Func7 (A, n − 2);
        return (s);
    end
end
```

Insertion into a Sorted Array

Input : Array A of n integers in sorted order.
$(\mathsf{A}[1] \leq \mathsf{A}[2] \leq \mathsf{A}[3] \ldots \leq \mathsf{A}[n])$ Element x.

function `SortedInsert`(A[ ],n,x)

$A[n+1] \leftarrow x$;

$j \leftarrow n$;

while $(j > 0)$ **and** $(A[j] > A[j+1])$ **do**

  `Swap`$(\mathsf{A}[j], \mathsf{A}[j+1])$;

  $j \leftarrow j - 1$;

end

`SortedInsert`: Example

Insert 12 in the following sorted array

$$[4, 7, 9, 14, 15, 17, 20, 25]$$

`SortedInsert`: Example

Insert x in the following sorted array

$$[4, 7, 9, 14, 15, 17, 20, 25]$$

In the worst case, how long does this take?

`SortedInsert`: Worst Case Running Time

Input : Array A of n integers in sorted order.
($\mathsf{A}[1] \leq \mathsf{A}[2] \leq \mathsf{A}[3] \ldots \leq \mathsf{A}[n]$) Element x.

function `SortedInsert(A[ ],n,x)`

$A[n+1] \leftarrow x$;

$j \leftarrow n$;

while $(j > 0)$ **and** $(A[j] > A[j+1])$ **do**

  `Swap`$(\mathsf{A}[j], \mathsf{A}[j+1])$;

  $j \leftarrow j - 1$;

end

`SortedInsert`: Example

Insert x in the following sorted array

$$[4, 7, 9, 14, 15, 17, 20, 25]$$

Assume x is equally likely to end up in any position in the array.

What is the expected time for this insertion?

`SortedInsert`: Expected Time

Input : Array A of n integers in sorted order.
$(\mathsf{A}[1] \leq \mathsf{A}[2] \leq \mathsf{A}[3] \ldots \leq \mathsf{A}[n])$ Element x.

function `SortedInsert`($\mathsf{A}[\ ]$,n,x)

$A[n+1] \leftarrow x$;

$j \leftarrow n$;

while $(j > 0)$ **and** $(A[j] > A[j+1])$ **do**

  `Swap`$(\mathsf{A}[j], \mathsf{A}[j+1])$;

  $j \leftarrow j - 1$;

end

Expected Running Time of `SortedInsert`

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

Insertion Sort

Input : Array A of n elements.
Result : Array A containing a permutation of the input such that $\mathsf{A}[1] \leq \mathsf{A}[2] \leq \mathsf{A}[3] \leq \ldots \leq \mathsf{A}[n]$.

`InsertionSort(A[ ],n)`

for $i \leftarrow 1$ **to** $n - 1$ **do**
  /* *insert* $\mathsf{A}[i + 1]$ *in* $\mathsf{A}[1..i]$ */
  /* *maintains:* $\mathsf{A}[1] \leq \mathsf{A}[2] \leq \mathsf{A}[3] \leq \ldots \leq \mathsf{A}[i]$ */
  $x \leftarrow A[i + 1]$;
  `SortedInsert(`A, i, x`)`;
end

`SortedInsert`: Example

Apply `InsertionSort` to the following array:

$$[4, 17, 7, 25, 15, 9, 20, 14, 12, 2, 19]$$

Insertion Sort: Worst Case Running Time

Input : Array A of n elements.
Result : Array A containing a permutation of the input such that $\mathsf{A}[1] \leq \mathsf{A}[2] \leq \mathsf{A}[3] \leq \ldots \leq \mathsf{A}[n]$.

`InsertionSort`$(\mathsf{A}[\,], n)$

for $i \leftarrow 1$ **to** $n - 1$ **do**
    /* *insert* $\mathsf{A}[i+1]$ *in* $\mathsf{A}[1..i]$ */
    /* *maintains:* $\mathsf{A}[1] \leq \mathsf{A}[2] \leq \mathsf{A}[3] \leq \ldots \leq \mathsf{A}[i]$ */
    $x \leftarrow A[i+1]$;
    `SortedInsert`$(A,\ i,\ x)$;
end

Insertion Sort: Expected Running Time

Input : Array A of n elements.
Result : Array A containing a permutation of the input such that $\mathsf{A}[1] \leq \mathsf{A}[2] \leq \mathsf{A}[3] \leq \ldots \leq \mathsf{A}[n]$.

```
InsertionSort(A[ ],n)
for i ← 1 to n − 1 do
    /* insert A[i + 1] in A[1..i]                                */
    /* maintains: A[1] ≤ A[2] ≤ A[3] ≤ . . . ≤ A[i]              */
    x ← A[i + 1];
    SortedInsert(A, i, x);
end
```

Insertion Sort (Version 2)

Input : Array A of n elements.
Result : Array A containing a permutation of the input such that $A[1] \leq A[2] \leq A[3] \leq \ldots \leq A[n]$.

Call to `SortedInsert` replaced by while loop.

```
InsertionSort(A[ ],n)
for i ← 2 to n do
    /* insert A[i] in A[1..(i − 1)]                          */
    /* maintains: A[1] ≤ A[2] ≤ A[3] ≤ ... ≤ A[i − 1]        */
    j ← i − 1;
    while (j > 0) and (A[j] > A[j + 1]) do
        Swap(A[j], A[j + 1]);
        j ← j − 1;
    end
end
```

Insertion Sort: Recursive Version

Input : Array A of n elements.

Result : Array A containing a permutation of the input such that $\mathsf{A}[1] \leq \mathsf{A}[2] \leq \mathsf{A}[3] \leq \ldots \leq \mathsf{A}[n]$.

```
InsertionSortRec(A[ ],n)
if (n > 1) then
    InsertionSortRec(A[ ],n − 1);
    /* Insert A[n] in A[1..(n − 1)]                */
    x ← A[n];
    SortedInsert(A, n − 1, x);
end
```

Insertion Sort: Recursive Version 2

Input : Array A of n elements.

Result : Array A containing a permutation of the input such that $A[1] \leq A[2] \leq A[3] \leq \ldots \leq A[n]$.

Call to `SortedInsert` replaced by while loop.

```
InsertionSortRec(A[ ],n)
if (n > 1) then
    InsertionSortRec(A[ ],n − 1);
    /* Insert A[n] in A[1..(n − 1)]                    */
    j ← n − 1;
    while (j > 0) and (A[j] > A[j + 1]) do
        Swap(A[j], A[j + 1]);
        j ← j − 1;
    end
end
```

Example

```
Func8(A, n)
/* A is an array of integers                                  */
if (n ≤ 2) then  return(A[1]) ;
else
    x ← 0;
    for i ← 1 to n − 1 do
        A[i] ← A[i] − A[i + 1];
        x ← x + A[i];
    end
    k ← Random(n − 1);
    x ← x+ Func8 (A, k);
    return (x);
end
```

Analysis of `Func8`

X_n = running time of **`Func8`** on array of size n.
$ET(n) = E(X_n)$ = expected running time of **`Func8`** array of size n.

$$\begin{aligned} ET(n) &= \sum_{q=1}^{n-1} \text{Prob}(k = q) E(X_n | k = q) \\ &= \sum_{q=1}^{n-1} \frac{1}{n-1}(cn + ET(q)) \\ &= ???. \end{aligned}$$

`Func8`: Upper Bounds

X_n = running time of **`Func8`** on array of size n.
$ET(n) = E(X_n)$ = expected running time of **`Func8`** on array of size n.

$$
\begin{aligned}
ET(n) &= E(X_n) \\
&= Pr(k \leq n/2)ET(X_n|k \leq n/2) + Pr(k > n/2)ET(X_n|k > n/2).
\end{aligned}
$$

$$
\begin{aligned}
Pr(k \leq n/2) &= 1/2. \\
Pr(k > n/2) &= 1/2. \\
ET(X_n|k \leq n/2) \leq ET(X_n|k = n/2) &= cn + ET(n/2). \\
ET(X_n|k > n/2) \leq ET(X_n|k = n-1) &= cn + ET(n-1).
\end{aligned}
$$

$$
\begin{aligned}
ET(n) &= Pr(k \leq n/2)ET(X_n|k \leq n/2) + Pr(k > n/2)ET(X_n|k > n/2) \\
&\leq \frac{1}{2}\Big(cn + ET(n/2)\Big) + \frac{1}{2}\Big(cn + ET(n-1)\Big) \\
&= cn + (1/2)ET(n/2) + (1/2)ET(n-1).
\end{aligned}
$$

`Func8`: Upper Bounds

$$
\begin{aligned}
ET(n) &\leq cn + (1/2)ET(n/2) + (1/2)ET(n-1) \\
&\leq cn + (1/2)ET(n/2) + (1/2)ET(n). \\
ET(n) - (1/2)ET(n) &\leq cn + (1/2)ET(n/2). \\
(1/2)ET(n) &\leq cn + (1/2)ET(n/2). \\
ET(n) &\leq 2cn + ET(n/2) \\
&\leq c_2 n + ET(n/2) \qquad \text{where } c_2 = 2c. \\
ET(n) &\leq c_2 n + c_2(n/2) + c_2(n/4) + c_2(n/8) + \ldots + c_2 \\
&= c_2 n(1 + (1/2) + (1/4) + (1/8) + \ldots + (1/n)) \\
&\leq c_2 n(1 + (1/2) + (1/4) + (1/8) + \ldots) = 2c_2 n.
\end{aligned}
$$

$$\therefore ET(n) \in O(n).$$

`Func8`: Lower Bounds

`Func8` always takes cn time, no matter what the value of k.
Therefore, $ET(n) \in \Omega(n)$.

Example

```
Func9(A, n)
/* A is an array of integers                    */
if (n ≤ 2) then  return(A[1]) ;
else
    x ← 0;
    for i ← 1 to n − 1 do
        A[i] ← A[i] − A[i + 1];
        x ← x + A[i];
    end
    k ← Random(⌊n/2⌋);
    x ← x+Func9 (A, k);
    x ← x+Func9 (A, n − k);
    return (x);
end
```

`Func9`: Analysis

X_n = running time of `Func9` on array of size n.
$ET(n) = E(X_n)$ = expected running time of `Func9` on array of size n.

$$\begin{aligned} ET(n) &= \sum_{q=1}^{\lfloor n/2 \rfloor} \text{Prob}(q = k) ET(X_n | k = q) \\ &= \sum_{q=1}^{\lfloor n/2 \rfloor} \frac{1}{n-1}(cn + ET(q) + ET(n-q)) \\ &= ???. \end{aligned}$$

`Func9`: Upper Bounds

X_n = running time of **`Func9`** on array of size n.
$ET(n) = E(X_n)$ = expected running time of **`Func9`** on array of size n.

$$\begin{aligned} ET(n) &= E(X_n) \\ &= Pr(k \leq n/4)ET(X_n|k \leq n/4) + Pr(k > n/4)ET(X_n|k > n/4). \end{aligned}$$

$$Pr(k \leq n/4) = 1/2.$$

$$Pr(k > n/4) = 1/2.$$

$$ET(X_n|k \leq n/4) \leq ET(X_n|k = 1) = cn + ET(n-1).$$

$$ET(X_n|k > n/4) \leq ET(X_n|k = n/4) = cn + ET(n/4) + ET(3n/4).$$

$\texttt{Func9}$: Upper Bounds

$$Pr(k \leq n/4) = 1/2.$$

$$Pr(k > n/4) = 1/2.$$

$$ET(X_n|k \leq n/4) \leq ET(X_n|k = 1) = cn + ET(n-1).$$

$$ET(X_n|k > n/4) \leq ET(X_n|k = n/4) = cn + ET(n/4) + ET(3n/4).$$

$$\begin{aligned}
ET(n) &= Pr(k \leq n/4)ET(X_n|k \leq n/4) + Pr(k > n/4)ET(X_n|k > n/4) \\
&\leq \frac{1}{2}\Big(cn + ET(1) + ET(n-1)\Big) + \frac{1}{2}\Big(cn + ET(n/4) + ET(3n/4)\Big) \\
&= cn + \frac{1}{2}\Big(c + ET(n-1)\Big) + \frac{1}{2}\Big(ET(n/4) + ET(3n/4)\Big).
\end{aligned}$$

`Func9`: Upper Bounds

$$
\begin{aligned}
ET(n) &= cn + \frac{1}{2}\Big(c + ET(n-1)\Big) + \frac{1}{2}\Big(ET(n/4) + ET(3n/4)\Big) \\
&\leq cn + \frac{1}{2}ET(n) + \frac{1}{2}\Big(ET(n/4) + ET(3n/4)\Big) \\
ET(n) - \frac{1}{2}ET(n) &\leq cn + \frac{1}{2}\Big(ET(n/4) + ET(3n/4)\Big). \\
\frac{1}{2}ET(n) &\leq cn + \frac{1}{2}\Big(ET(n/4) + ET(3n/4)\Big). \\
ET(n) &\leq 2cn + ET(n/4) + ET(3n/4) \\
&\leq c'n + ET(n/4) + ET(3n/4) \qquad \text{where } c' = 2c.
\end{aligned}
$$

$$\therefore ET(n) \in O(n \log_2(n)).$$

`Func9`: Lower Bounds

In the best case, $k = n/2$.

$$
\begin{aligned}
ET(n) &\geq cn + ET(n/2) + ET(n/2) \\
&= cn + 2ET(n/2). \\
\therefore\ ET(n) &\in \Omega(n \log_2(n)).
\end{aligned}
$$