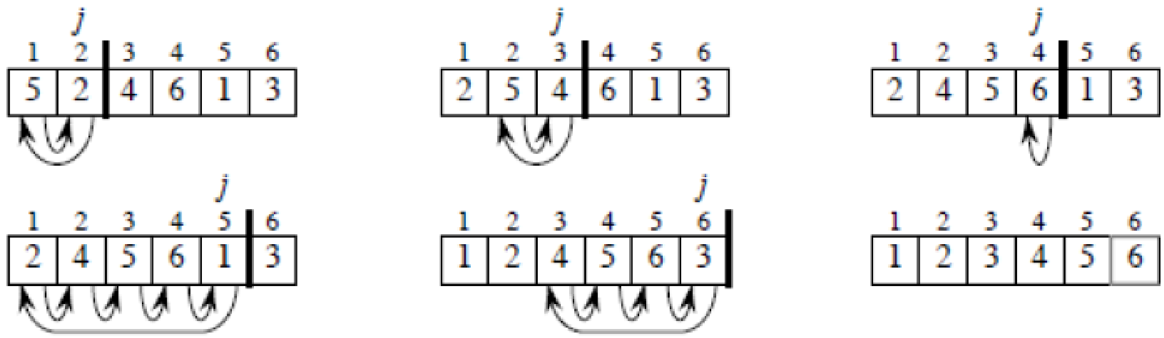
1-a

Example:



1-b

- Assume that the i th line takes time c_i , which is a constant. (Since the third line is a comment, it takes no time.)
- For j = 2, 3,... , n, let t_j be the number of times that the while loop test is executed for that value of j .
- Note that when a for or while loop exits in the usual way due to the test in the loop header the test is executed one time more than the loop body.

The running time of the algorithm is

$$\sum_{\text{all statements}} (\text{cost of statement}) \cdot (\text{number of times statement is executed}) .$$

Let *T (n)* = running time of I insertion sort T .

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) \\ &\quad + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) . \end{aligned}$$

The running time depends on the values of t_j . These vary according to the input.

Best case: The array is already sorted.

- Always find that $A[i] \le key$ upon the first time the **while** loop test is run (when $i = j - 1$).
- All t_j are 1.
- Running time is
$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) . \end{aligned}$$
- Can express $T(n)$ as $an+b$ for constants a and b (that depend on the statement costs c_i) $\Rightarrow T(n)$ is a *linear function* of n.

1-c

Worst case: The array is in reverse sorted order.

- Always find that $A[i] > key$ in while loop test.
- Have to compare *key* with all elements to the left of the jth position $\Rightarrow$ compare with $j-1$ elements.
- Since the while loop exits because i reaches 0, there's one additional test after the $j-1$ tests $\Rightarrow t_j = j$.
- $\displaystyle\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} j$ and $\displaystyle\sum_{j=2}^{n} (t_j - 1) = \sum_{j=2}^{n} (j-1)$.
- $\displaystyle\sum_{j=1}^{n} j$ is known as an ***arithmetic series***, and equation (A.1) shows that it equals $\dfrac{n(n+1)}{2}$.
- Since $\displaystyle\sum_{j=2}^{n} j = \left(\sum_{j=1}^{n} j\right) - 1$, it equals $\dfrac{n(n+1)}{2} - 1$.
- Letting $k = j-1$, we see that $\displaystyle\sum_{j=2}^{n} (j-1) = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2}$.
- Running time is

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\ &\quad - (c_2 + c_4 + c_5 + c_8)\,. \end{aligned}$$

- Can express $T(n)$ as $an^2 + bn + c$ for constants a, b, c (that again depend on statement costs) $\Rightarrow T(n)$ is a *quadratic function* of n.

- Running time is

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\ &\quad - (c_2 + c_4 + c_5 + c_8)\,. \end{aligned}$$

- Can express $T(n)$ as $an^2 + bn + c$ for constants a, b, c (that again depend on statement costs) $\Rightarrow T(n)$ is a *quadratic function* of n.

1.d

Initialization: Just before the first iteration, $j = 2$. The subarray $A[1 .. j - 1]$ is the single element $A[1]$, which is the element originally in $A[1]$, and it is trivially sorted.

Maintenance: To be precise, we would need to state and prove a loop invariant for the "inner" **while** loop. Rather than getting bogged down in another loop invariant, we instead note that the body of the inner **while** loop works by moving $A[j - 1]$, $A[j - 2]$, $A[j - 3]$, and so on, by one position to the right until the proper position for *key* (which has the value that started out in $A[j]$) is found. At that point, the value of *key* is placed into this position.

Termination: The outer **for** loop ends when $j > n$; this occurs when $j = n + 1$. Therefore, $j - 1 = n$. Plugging n in for $j - 1$ in the loop invariant, the subarray $A[1 .. n]$ consists of the elements originally in $A[1 .. n]$ but in sorted order. In other words, the entire array is sorted!

2.

$(\lg n)!$

$\sqrt{2}^{\lg n}$

$2^{\sqrt{2 \lg n}}$

$(\lg n)! = \omega(n^3)$ by taking logs: $\lg(\lg n)! = \Theta(\lg n \lg \lg n)$ by stirling's approx., $\lg(n^3) = 3\lg n$, $\lg\lg n = \omega(3)$

or

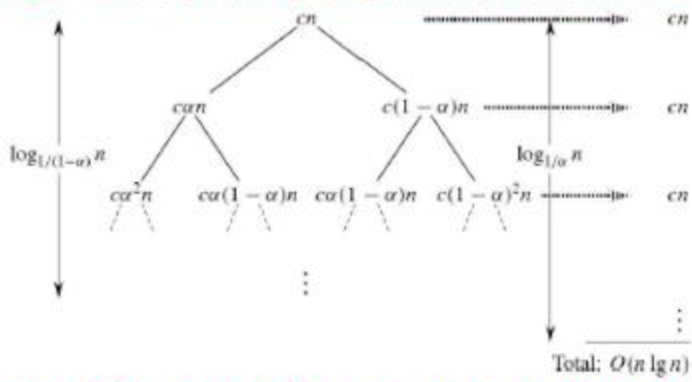
$(\lg n)! = \Theta((\lg n)^{\lg n + \frac{1}{2}} e^{-\lg n})$ by substituting $\lg n$ for n in the previous justification, $(\lg n)! = \Theta((\lg n)^{\lg n + 1/2} n^{-\lg e})$ because $a^{\log_b c} = c^{\log_b a}$

$(\sqrt{2})^{\lg n} = \sqrt{n}$ because $(\sqrt{2})^{\lg n} = 2^{(1/2)\lg n} = 2^{\lg \sqrt{n}} = \sqrt{n}$

$2^{\sqrt{2 \lg n}} = n^{\sqrt{2/\lg n}}$ by raising identity 4 to the power $\sqrt{2 \lg n}$

3.

Assume that $\alpha \geq 1 - \alpha$, then by translating the equation $T(n)$ to a recursion tree, we have:



Since the shorter part of recursion tree is $\log_{\frac{1}{1-\alpha}} n$, we can guess the lower bound is:

$$T(n) = \Omega\left(n \log_{\frac{1}{1-\alpha}} n\right) = \Omega(n \log_2 n)$$

, and the taller part of the tree is $\log_{\frac{1}{\alpha}} n$, we can guess the upper bound is:

$$T(n) = O(\log_{\frac{1}{\alpha}} n) = O(n \log_2 n)$$

First, we prove the upper bound:
Assume that $T(n) \leq dn \log_2 n$ for a constant $d > 0$. Substitute $T(n) \leq dn \log_2 n$ into the equation, we have:

$$\begin{aligned}
T(n) &= T(\alpha n) + T\big((1-\alpha)n\big) + cn \\
&\leq d\alpha n \log_2(\alpha n) + d(1 \\
&\quad - \alpha)n \log_2\big((1-\alpha)n\big) + cn \\
&= d\alpha n \log_2 \alpha + d\alpha n \log_2 n \\
&\quad + d(1-\alpha)n \log_2(1-\alpha) + d(1-\alpha)n \log_2 n + cn \\
&= dn \log_2 n + dn(\alpha \log_2 \alpha + (1-\alpha)\log_2(1-\alpha)) + cn \\
T(n) \leq dn \log_2 n &\qquad \text{if } dn(\alpha \log_2 \alpha + (1-\alpha)\log_2(1-\alpha)) + cn \leq 0
\end{aligned}$$

, then $d(\alpha \log_2 \alpha + (1-\alpha)\log_2(1-\alpha)) \leq -c$.
Since $\alpha \geq 1 - \alpha$, $1 > \alpha \geq 1/2$, $\log_2 \alpha < 0$ and $\log_2(1-\alpha) < 0$, thus, $\alpha \log_2 \alpha + (1 - \alpha)\log_2(1-\alpha)) < 0$, so we get the condition of constant d as:

$$d \geq \frac{-c}{\alpha \log_2 \alpha + (1-\alpha)\log_2(1-\alpha))} \text{ or } d \geq \frac{c}{-\alpha \log_2 \alpha - (1-\alpha)\log_2(1-\alpha))}$$

With the same proof of uppers bound, by substituting $T(n) \geq dn \log_2 n$, we can prove the lower bound $T(n) = \Omega(n \log_2 n)$, if $0 < d \leq \frac{c}{-\alpha \log_2 \alpha - (1-\alpha)\log_2(1-\alpha))}$

4. 1times = 1/n

Since Hire-assistant always hires candidate 1, it hires exactly once if and only if no candidate other than candidate 1 are hired. This event occurs when candidate 1 is the best candidate of the n, which occur with probability 1/n.

ntimes = 1/n!

Hire assistant hires ntimes if each candidate is better than all those who were interviewed (and hired) before. This event occurs precisely when the list of ranks given to the algorithms is $\langle 1, 2, \cdots, n \rangle$, which occurs with probability of $1/n!$.

2times

In order fore Hire-assistant to hire exactly twice candidate 1 musthave rank $i \le n-1$ and all candidates whose ranks are $i+1, i+2 \ldots n-1$ must be interviewed after the candidate whose rank is n.
Let E_i be the event in which candidate 1 has rank i; clearly, $\Pr\{E_i\} = 1/n$ for any given value of i.
Leting j denote the position in the interview order of the best candidate, let F be the event in which candidates $2, 3 \cdots, j-1$ have ranks strictly less than the rank of candidate 1. Given that event E_i has occured, event F occurs when the best $i+1, i+2, \cdots n$. Thus, $\Pr\{F \mid E_i\} = 1/(n-i)$
Our final event is A, which occurs when Hire-assistant hires exactly twice. Nothing that the events $E_1, E_2 \cdots E_n$ are disjoint, we have

$$A = F \cap (E_1 \cup E_2 \cup \cdots \cup E_{n-1})$$
$$= (F \cap E_1) \cup (F \cup E_2) \cup \cdots \cup (F \cap E_{n-1}).$$

and
$$\Pr\{A\} = \sum_{i=1}^{n-1} \Pr\{F \cap E_i\}$$

By equation (C.14), $\Pr\{F \cap E_i\} = \Pr\{F \mid E_i\} \Pr\{E_i\} = \frac{1}{n-i} \cdot \frac{1}{n}$

and so
$$\Pr\{A\} = \sum_{i=1}^{n-1} \frac{1}{n-i} \cdot \frac{1}{n} = \frac{1}{n} \sum_{i=1}^{n-1} \frac{1}{n-i} = \frac{1}{n}\left(\frac{1}{n-1} + \frac{1}{n-2} + \cdots + \frac{1}{1}\right) = \frac{1}{n} \cdot H_{n-1}$$

nth harmonic number.