CSE331 Introduction to Algorithm
Lecture 11: The Hiring Problem

Antoine Vigneron
antoine@unist.ac.kr

Ulsan National Institute of Science and Technology

July 23, 2021

Introduction

- This lecture is an introduction to *randomized* algorithms.
- We study efficient randomized algorithms for the *hiring problem*.
- Some of the proofs will differ from the textbook.
- Reference: Section 5.1 of the textbook Introduction to Algorithms by Cormen, Leiserson, Rivest and Stein. (Available online from the UNIST library website.)

The Hiring Problem

- You want to hire a new assistant.
- An employment agency sends you a candidate to interview every day.
- If the candidate is better than your previous assistant, you replace the previous assistant with the new candidate.

Algorithm

```
procedure HireAssistant(n)
    best ← 0                                  ▷ Least qualified dummy candidate
    for i ← 1, n do
        interview candidate i
        if candidate i is better than candidate best then
            best ← i
            hire candidate i
```

The Hiring Problem

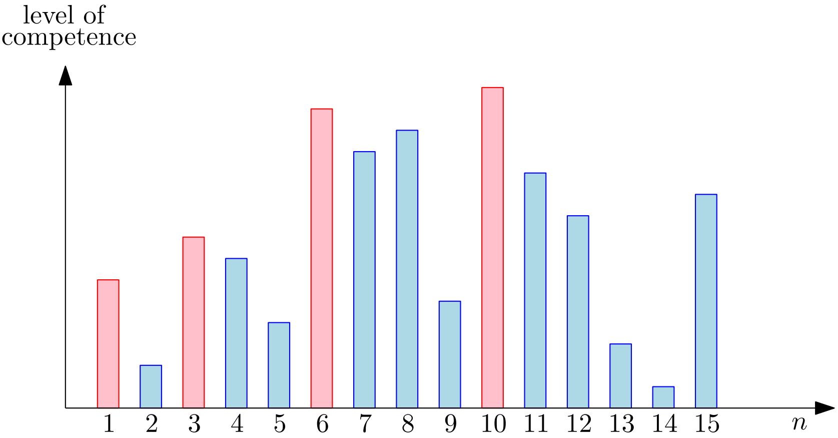
- Interviewing one candidate costs you c_i, hiring a new employee has a much higher cost c_h.
- So total cost is $nc_i + mc_h$ where m is the number of times we hire.

Problem

How can we minimize the total cost?

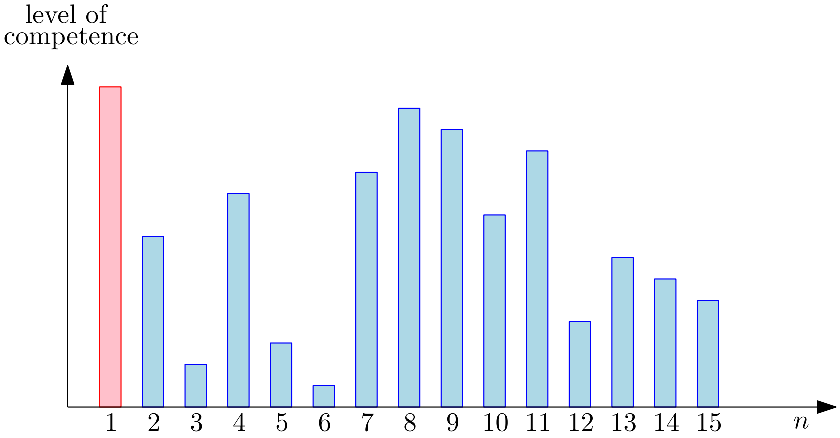
- As the term nc_i only depends on n, we focus on the hiring cost mc_h.
- In other words, we want to minimize m, the number of times we hire.

Example 1



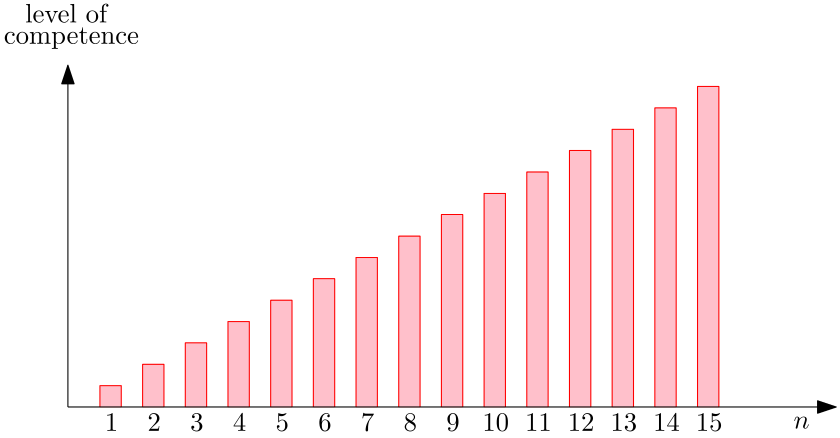
- If candidates are interviewed in this order, we hire 4 times: $m = 4$.

Example 2



- Best case: we only hire the first applicant, and $m = 1$.

Example 3



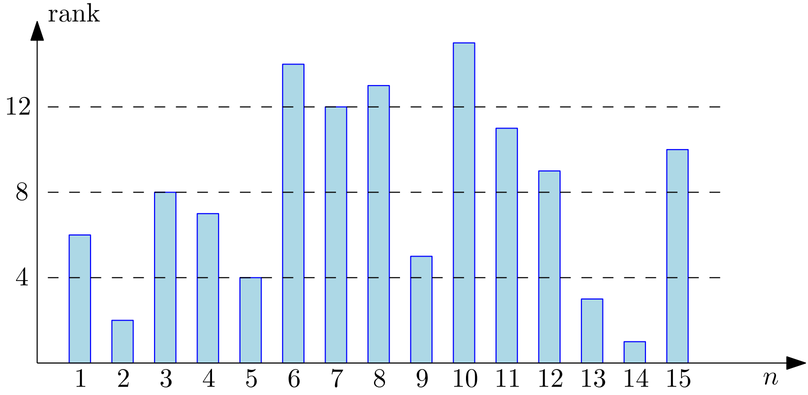
- Worst case: $m = n$.

Assumption

- We made the implicit assumption that candidates can be ranked.
- So each candidate i has a *rank* r_i, and $r_i > r_j$ means that r_i is better qualified.
- Without loss of generality, we may assume that $(r_1, \dots, r_n)$ is a permutation of $(1, \dots, n)$.
- So we hire candidate i iff

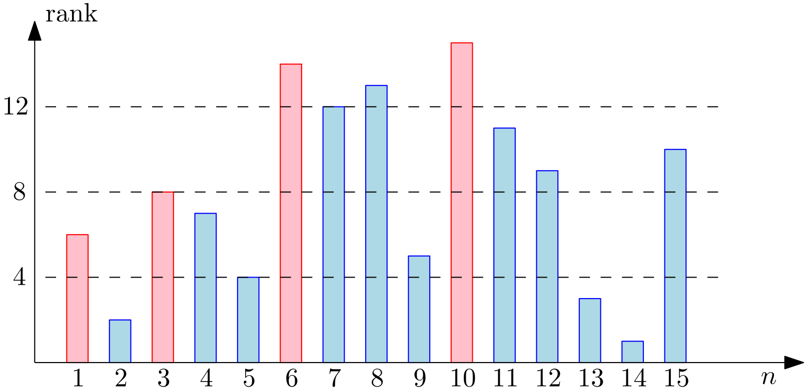
$$r_i = \max_{j=1,\dots,i} r_j.$$

Example



- $(r_1, \ldots, r_n) = (6, 2, 8, 7, 4, 14, 12, 13, 5, 15, 11, 9, 3, 1, 10)$

Example



- $(r_1, \ldots, r_n) = (6, 2, 8, 7, 4, 14, 12, 13, 5, 15, 11, 9, 3, 1, 10)$

Analysis

- Before the interviews, we do not know the ranks.
- So we make the assumption that candidates arrive in a *random order* : $(r_1, \dots, r_n)$ is a permutation of $(1, \dots, n)$ chosen uniformly at random.
- Let X be the random variable whose value is the number of times we hire.
- Then the expected hiring cost in our process is

$$E[Xc_h] = E[X]c_h.$$

- So in order to analyze this algorithm, we will estimate $E[X]$.

Analysis

- We define the random variable X_i as follows:

$$X_i = \begin{cases} 1 & \text{if candidate } i \text{ is hired,} \\ 0 & \text{otherwise.} \end{cases}$$

- Then

$$X = \sum_{i=1}^{n} X_i.$$

Analysis

Definition

The *indicator random variable* I_A associated with event A is the random variable that takes value 1 if A occurs, and 0 otherwise.

Example

X_i is the indicator random variable associated with the event *candidate i is hired*.

Proposition

$E[I_A] = \Pr(I_A = 1)$ *for any indicator random variable.*

- Proof:

$$\begin{aligned} E[I_A] &= 0 \cdot \Pr(I_A = 0) + 1 \cdot \Pr(I_A = 1) \\ &= \Pr(I_A = 1) \end{aligned}$$

Analysis

Theorem (Linearity of Expectation)

Let $Y_1, \ldots, Y_n$ be n random variables. Let $Y = \sum_{i=1}^{n} Y_i$. Then

$$E[Y] = E\left[\sum_{i=1}^{n} Y_i\right] = \sum_{i=1}^{n} E[Y_i].$$

- Remark: It holds for any random variables, even if they are not independent. See discrete maths or probability course.

Analysis

- Going back to our analysis:

$$
\begin{aligned}
E[X] &= E\left[\sum_{i=1}^{n} X_i\right] \\
&= \sum_{i=1}^{n} E[X_i] && \text{by linearity of expectation} \\
&= \sum_{i=1}^{n} \Pr[X_i = 1] && \text{because } X_i \text{ is an indicator variable} \\
&= \sum_{i=1}^{n} \Pr[\text{candidate } i \text{ is hired}]
\end{aligned}
$$

Analysis

- We still need to determine Pr[candidate i is hired].
- It is the probability that r_i is the largest rank among $r_1, r_2 \ldots, r_i$.
- As $(r_1, \ldots, r_n)$ is a permutation of $(1, \ldots, n)$ chosen uniformly at random, this probability is $1/i$.
- Therefore
$$E[X] = \sum_{i=1}^{n} \frac{1}{i}.$$
- This is the *harmonic series*, and it is well known that
$$\sum_{i=1}^{n} \frac{1}{i} = \ln(n) + O(1).$$

Analysis

Theorem

If the candidates are presented in a random order, HIREASSISTANT *has an average-case hiring cost* $\Theta(c_h \log n)$.

- This is much better than the worst case $c_h n$.
- Problem: If the candidates do not appear in random order, it does not hold. (For instance if they appear by decreasing rank.)

How to Fix It?

- Suppose that the employment agency sends you in advance a list of n candidates, and every week you can choose which candidate you interview.
- You can first compute a random permutation of this list, and then interview in this order.
- Then the expected hiring cost becomes $\Theta(c_h \log n)$ regardless of the order of the original list.

Randomized Algorithm

Pseudocode

procedure RandomizedHireAssistant(n)
  Randomly permute the list of candidates
  $best \leftarrow 0$
  for $i \leftarrow 1, n$ **do**
    interview candidate i
    if candidate i is better than candidate $best$ **then**
      $best \leftarrow i$
      hire candidate i

Theorem

The expected hiring cost of RandomizedHireAssistant *is* $\Theta(c_h \log n)$.

Computing a Random Permutation

Pseudocode

procedure PermuteBySorting($A[1 \dots n]$)
  $P[1 \dots n] \leftarrow$ new array
  for $i \leftarrow 1, n$ **do**
    $P[i] \leftarrow \text{random}(1, n^3)$ ▷ random number in $\{1, 2, \dots, n^3\}$
  Sort A using $P[i]$ as the key of $A[i]$ for all i

- Problem: if two keys are equal, it fails, i.e. the permutation is not chosen uniformly at random.
- The random number is chosen in $\{1, \dots, n^3\}$ to ensure that it happens with probability $\leqslant 1/n$. (left as an exercise).
- In practice just generate a random floating-point number in $[0, 1)$.

Computing a Random Permutation

- Remark: This method can be used with software such as Microsoft Excel or Open Office: generate a column of random numbers and sort according to it.

Theorem

If all keys are distinct, then PermuteBySorting *produces a uniform random permutation.*

- Proof in textbook (Lemma 5.4 p. 125), not covered.
- Other problems:
  - this algorithm is not *in place* because it uses the auxiliary array $P[1 \ldots n]$.
  - It runs in $\Theta(n \log n)$ time if we sort using Merge sort.
- How to fix it?

In-Place Computation of a Random Permutation

Pseudocode

procedure RandomizeInPlace($A[1 \dots n]$)
    for $i \leftarrow 1, n-1$ **do**
        Exchange $A[i]$ with $A[\text{random}(i, n)]$
                        ▷ random number in $\{i, \dots, n\}$

Theorem

RandomizeInPlace *computes a uniform random permutation.*

- Proof done in class, see textbook p. 126.

Concluding Remarks

- In summary, for this problem, we can avoid worst-case behavior by computing a random permutation of the input.
- This approach applies to many other problems. For instance geometric problems. (See CSE520 Computational geometry.)
- Another example in next lecture (QUICKSORT).