Introduction to Algorithms and Data Structures

Lesson 2: Foundation of Algorithms (2) Simple Basic Algorithms

Professor Ryuhei Uehara,
School of Information Science, JAIST, Japan.
uehara@jaist.ac.jp
http://www.jaist.ac.jp/~uehara

Algorithm?

- Algorithm: abstract description of how to solve a problem (by computer)
  - It returns correct answer for any input
  - It halts for any input
  - Description is not ambiguity
    - (operations are well defined)

- Program: description of algorithm by some computer language
  - (Sometimes it never halt)

Al-Khwarizmi

Design of Good Algorithm

- There are some design method
- Estimate time complexity (running time) and space complexity (quantity of memory)
- Verification and Proof of Correctness of Algorithm

- Bad algorithm
  - Instant idea: No design method
  - Just made it: No analysis of correctness and/or complexity

Simple example and algorithm

- Stock trading algorithm

  Goal: Maximize your benefit

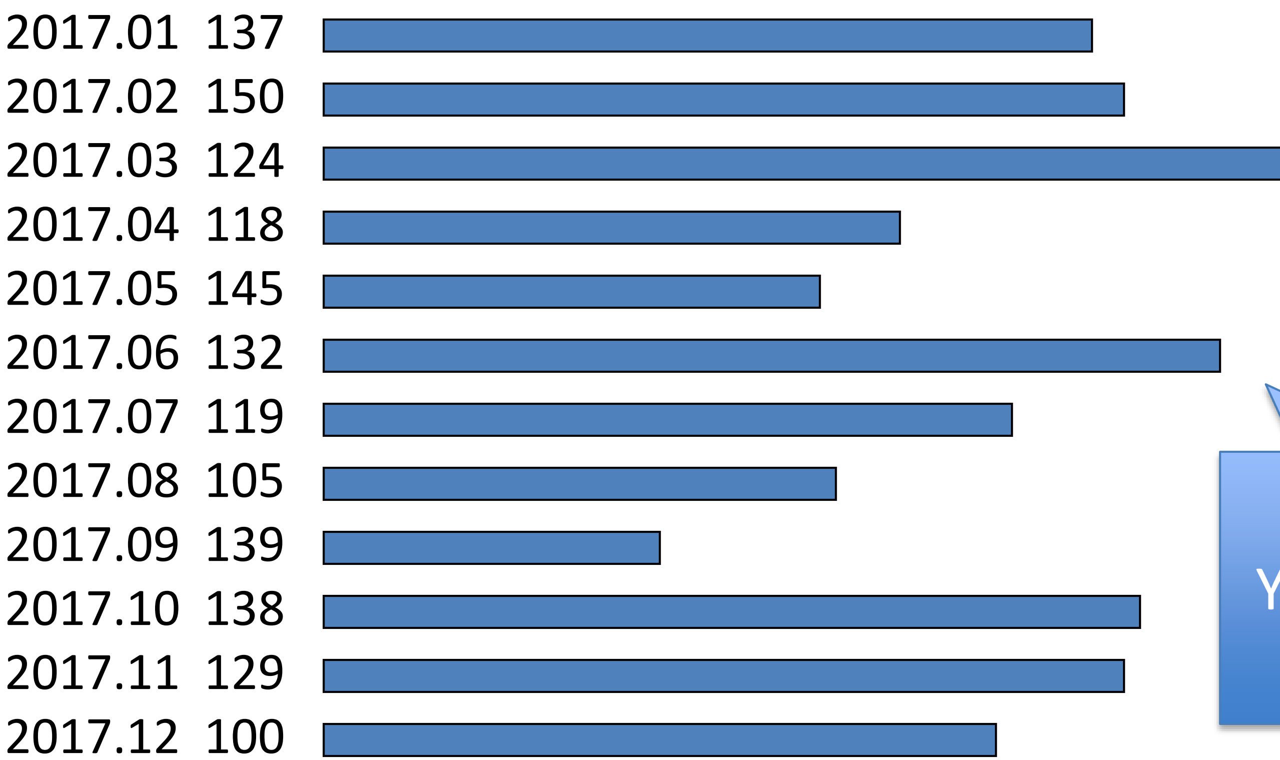
  – Naïve method

  – Some improvements

  – More improvement: from O(n^2) to O(n)

Stok trading (maximize benefit)

- You would buy once and sell once. Can you find the <u>maximum</u> benefit?



Note:
You cannot sell before buy!!

Formalization of the problem

- int sp[n]: array of stock prices (e.g. n=12)
- When you buy at month i and sell at month j
  - buy: sp[i]
  - sell: sp[j]
  - benefit: sp[j] - sp[i]
- Goal: maximize sp[j]-sp[i]

  That is, compute the following;

  max{sp[j] - sp[i] | 0<= i < j < n}

Outline of algorithms

- Method A

```
for i=0 to n-2
  for j=i+1 to n-1
    find benefit sp[j]-sp[i]
```

- Method B:

```
for j=1 to n-1
  for i=0 to j-1
    find benefit sp[j]-sp[i]
```

Algorithm based on method A

- Is the following algorithm efficient?

```
MaxBenefit(sp[],n){/*sp[0]…sp[n-1]*/
  mxp=0; /*Maximum benefit*/
  for i=0 to n-2
    for j=i+1 to n-1
      d = sp[j] – sp[i]; /*benefit*/
      if d > mxp then mxp = d;
                  /*Update max. benefit*/
    endfor
  endfor
  return mxp;
}
```

Algorithm based on method A

- Is the following algorithm efficient?

```
MaxBenefit(sp[],n){/*sp[0]…sp[n-1]*/
  mxp=0; /*Maximum benefit*/
  for i=0 to n-2
    for j=i+1 to n-1
      d = sp[j] – sp[i]; /*benefit*/
      if d > mxp then mxp = d;
                /*Update max. benefit*/
    endfor
  endfor
  return mxp;
}
```

For fixed i, benefit is maximum when sp[j] is maximum
→ We don't need to compute sp[j]-sp[i] every time

Algorithm based on method A (Improved)

```
MaxBenefit(sp[],n){ /*sp[0]…sp[n-1]*/
  mxp=0; /* Maximum benefit */
  for i=0 to n-2
    mxsp = sp[i];
    for j=i+1 to n-1
      if sp[j] > mxsp then mxsp = sp[j];
    endfor
    d = mxsp – sp[i];
    if d > mxp then mxp = d;
  endfor
  return mxp;
}
```

mxsp: maximum trade

Subtraction is out of loop

Outline of algorithms

- Method A

```
for i=0 to n-2
  for j=i+1 to n-1
    find benefit sp[j]-sp[i]
```

- Method B:

```
for j=1 to n-1
  for i=0 to j-1
    find benefit sp[j]-sp[i]
```

Algorithm based on method B

```
MaxBenefit(sp[],n){ /*sp[0]…sp[n-1]*/
  mxp=0; /* Maximum benefit */
  for j=1 to n-1
    mnsp = sp[j];
    for i=0 to j-1
      if sp[i] < mnsp then mnsp = sp[i];
    endfor
    d = sp[j] - mnsp;
    if d > mxp then mxp = d;
  endfor
  return mxp;
}
```

mnsp: cheapest stock price

Efficiency of algorithms

- Number of loops (or repeating)
  - Method (A): number of loops is O(n^2)

$$\sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} (n-1-i) = \frac{n^2-n}{2} \leq n^2/2$$

Maybe tomorrow?

  - Method (B): number of loops is O(n^2)

$$\sum_{j=1}^{n-1} \sum_{i=0}^{j-1} 1 = \sum_{j=1}^{n-1} j = \frac{n^2-n}{2} \leq n^2/2$$

Notation that proportion to n^2

Q. Can we decrease them?

More improvement of algorithms; decreasing the number of loops

- Consider the second loop
  - Method A:
    - `MAX[i,n-1]` is the maximum between time `i` and time `n-1`
    - It computes in order `MAX[1,n-1]`, `MAX[2,n-1]`,…

    Q: can we compute `MAX[i,n-1]` from `MAX[i-1,n-1]`?

    NO!
  - Method B:
    - `MIN[0,j-1]` is the minimum between time `0` to time `j-1`
    - It computes in order `MIN[0,0]`, `MIN[0,1]`, …

    Q: can we compute `MIN[0,j]` from `MIN[0,j-1]`?

    YES! `MIN[0,j] = min(MIN[0,j-1],sp[j])`

Algorithm based on method B

```
MaxBenefit(sp[],n){ /*sp[0]…
  mxp=0; /* Maximum benefit
  for j=1 to n-1
    mnsp = sp[j];
    for i=0 to j-1
      if sp[i] < mnsp then mnsp = sp[i];
    endfor
    d = sp[j] - mnsp;
    if d > mxp then mxp = d;
  endfor
  return mxp;
}
```

- When j=k:
  mnsp is the minimum between sp[0] to sp[k-1]
- When j=k+1:
  mnsp is the minimum between sp[0] to sp[k]

We can keep msf, the minimum when j=k, and use it; when j=k+1, the minimum is the smaller one of msf and sp[k].

Efficient algorithm

- Algorithm that runs in O(*n*) time

```
MaxBenefit(sp[],n){ /*sp[0]…sp[n-1]*/
  mxp=0; /* Maximum benefit */
  msf = sp[0]; /* Cheapest value so far */
  for j=1 to n-1
    d = sp[j] - msf;
    if d > mxp then mxp = d;
    if sp[j] < msf then msf = sp[j];
  endfor
  return mxp;
}
```