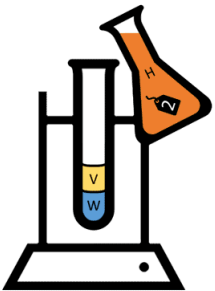
T193 Liquid Layers



Written by Alex Tung
Prepared by Percy Wong

4 May 2019

Statistics

Attempts	Max	Mean	Std Dev	Subtasks
40	62.985	31.007	20.705	100: 0 62.985: 1 55.238: 1 48: 10 44.905: 1 44.896: 1 44.872: 1 44.864: 1 44.857: 1 44.829: 1 44.821: 2 44.816: 1 43.533: 1 43.249: 1 35.712: 1 34.95: 1 11.189: 4 11.179: 1 5: 5

Highest score: 62.985 (cost = 1078) by `wjx` at 4:48

Problem Remodel

- test tube: a natural ”machine” to sort the liquids by density (high to low)
- machine counter: what is it counting?

Problem Remodel

- test tube: a natural ”machine” to sort the liquids by density (high to low)
- machine counter: what is it counting?
  - the machine sorts the liquids whenever you pour a new one
  - test tube: [7, 5, 3, 2, 1], `addLiquid(4)`
  - test tube: [7, 5, **4**, 3, 2, 1], `counter += 3`

Problem Remodel

- test tube: a natural ”machine” to sort the liquids by density (high to low)
- machine counter: what is it counting?
  - the machine sorts the liquids whenever you pour a new one
  - test tube: [7, 5, 3, 2, 1], `addLiquid(4)`
  - test tube: [7, 5, **4**, 3, 2, 1], `counter += 3`
- machine counter: NUMBER OF INVERSIONS! (in reverse order)

Number of inversion of an array $a[]$ is the number of ordered tuples (i, j) such that $i < j$ and $a[i] > a[j]$.

M1753 Number of Inversions

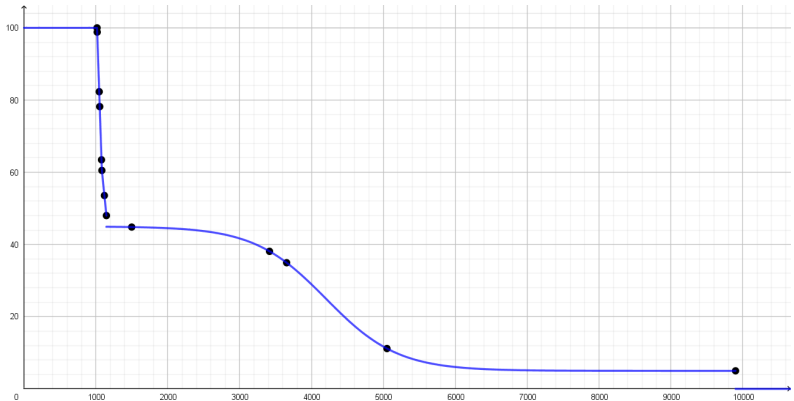
Problem Remodel

- query the number of inversions of a sequence of items S
- cost of a query is $length(S)$
- have to find the correct order of $[item_1, item_2, \cdots, item_N]$

Scoring

$$score = \begin{cases} 100, & \text{if } C \leq 1016 \\ 60 + 40 \times \frac{1083 - C}{1083 - 1016}, & \text{if } 1016 < C \leq 1083 \\ 48 + 12 \times \frac{1146 - C}{1146 - 1083}, & \text{if } 1083 < C \leq 1146 \\ 5 + 40 \div (1 + e^{(\frac{C - 4200}{500})}), & \text{if } 1146 < C \leq 9900 \\ 0, & \text{if } C > 9900 \end{cases}$$

Scoring


100
80
60
40
20
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

Adaptive Grader

In some test cases the behavior of the grader is adaptive. This means that in these test cases the grader does not have a fixed order of density of the N liquids. Instead, the answers given by the grader may depend on the questions asked by your solution.

- Idea by Tony Wong and Anson Ho
- Do you have any idea on its algorithm?

Compare Function of Two Elements

```
bool cmp(int u, int v) {
    pourLiquid(u);
    pourLiquid(v);
    return getReading() == 0;
}
```

- `true` if `u` should be placed in front of `v`
- `false` if `v` should be placed in front of `u`

Basic Solutions - Bubble Sort

```
for (int i = 0; i < N; i++)
    order[i] = i + 1; // initialize
for (int i = 0; i + 1 < N; i++)
    for (int j = 0; j + 1 < N - i; j++)
        if (cmp(order[j + 1], order[j]))
            swap(order[j + 1], order[j]);
```

- Called `cmp(u, v)` for $N(N-1)/2$ times
- Cost $= N(N-1) = 9900$, score $= 5.000(448...)$

Basic Solutions - Improved Insertion Sort

```
for (int i = 1; i < N; i++) {
    for (int j = 0; j <= i; j++)
        pourLiquid(order[j]);
    int verdict = getReading(); // no. of inversions
    for (int j = 0; j < verdict; j++)
        swap(order[i - j], order[i - j - 1]);
}
```

- Called `pourLiquid(index)` for $2 + 3 + \cdots + N$ times
- Cost $= (N + 2)(N - 1)/2 = 5049$, score $= 11.189$

Basic Solutions

- Merge Sort
  - Cost $= 1146$
  - Score $= 48.000$
- Insertion Sort with Binary Search
  - Cost $= 1146$
  - Score $= 48.000$

Basic Solutions - Standard Library

"Algorithm"	Cost	Score
`sort(order, order + N, cmp)`	3654	34.950
`sort(order, order + N, cmp)` with cache	3416	38.100
`stable_sort(order, order + N, cmp)`	1498	44.821
`stable_sort(order, order + N, cmp)` with cache	1358	44.864
`qsort(order, N, sizeof(int), ___)`	1146	48.000

Note: implementation of `qsort` on GCC is actually a merge sort

Optimization - Improved Merge Sort

- If only one item is left in one of the two lists, switch to Insertion Sort
- If only one item is left in one of the two lists and
  two items are left in the another list,
  then their order can be determined by one query with cost $= 3$

Optimization - Ternary Search

- Recall Improved Insertion Sort, it is possible to determine the position of a new item in a sorted sequence S' with cost $= length(S') + 1$
- Each query in Binary Search: $length(S') = 1$, cost $= 2$
  effectiveness $= \log 2/2 = 0.5$
- Each query in Ternary Search: $length(S') = 2$, cost $= 3$
  effectiveness $= \log 3/3 \approx 0.528$
- Effectiveness of n-ary search ≤ 0.5 for $n \geq 4$
- Use dynamic programming to decide whether Binary Search or Ternary Search should be used for each $length(S')$

Optimization - Summary

	Cost	Score
MS	1146	48.000
MS with insertion by BS	1146	48.000
MS with insertion by BS and cost 3 query on $1 + 2$ case	1118	53.333
MS with insertion by BS and TS	1083	60.000

MS: Merge Sort
BS: Binary Search
TS: Ternary Search

Optimization - Summary

	Cost	Score
IS with BS	1146	48.000
IS with BS and cost 3 query on $1+2$ case	1083	60.000
IS with TS	1078	62.985
IS with BS and TS	1046	82.090

IS: Insertion Sort
BS: Binary Search
TS: Ternary Search

Basic Solutions - Merge Insertion Sort

- Main idea
  1. Divide a sequence of length k into $k/2$ pairs of items
  2. Recursively sort the items with greater value in each pair
  3. Insert the items with smaller value in each pair to the sorted sequence by Binary Search with some special order
- `https://en.wikipedia.org/wiki/Merge-insertion_sort`
  - Exercise: find a mistake in the page (rev: 16 Aug 2018)
- It is the best known polynomial time comparison sort in terms of minimizing the number of comparison in worst case
- Cost $= 1068$, score $= 68.955$

Limitation of Comparison Sort

- cost$(N) = 2f(N)$ where $f(N)$ is the minimal number of comparisons needed to sort n elements (OEIS A036604)
- cost$(100) = 2f(100) \leq 1068$ (Merge Insertion Sort)
- cost$(100) = 2f(100) \geq 2\lceil \log 100! \rceil = 1050$
  (assume each comparison eliminates half of the possibilities)
- Therefore, Comparison Sort is not enough to get full score even if you find the exact value of $f(100)$, which is an unsolved problem

Optimization - Improved Merge Insertion Sort

- Merge Insertion Sort is optimized for minimizing the number of comparisons (a.k.a. query with cost $= 2$) only
- Possible improvements with dynamic programming
  - Re-optimize the insertion order
  - Consider dividing into two halves with unequal sizes

Optimization - Summary

	Cost	Score
MIS with BS	1068	68.955
MIS with BS and OPT	1068	68.955
MIS with BS and TS	1018	98.806
MIS with BS, TS and OPT	1018	98.806

MIS: Merge Insertion Sort
BS: Binary Search
TS: Ternary Search
OPT: Consider dividing into two halves with unequal sizes

Full solution

- Let $k = length(S)$
- Number of possibilities of number of inversions $= k(k-1)/2 + 1$
- Upper bound of effectiveness $= \log[k(k-1)/2 + 1]/k$
  which attains its maximum at $k = 5$
- It suggests that queries with $k = 4, 5$ may be more effective
- Unfortunately (or fortunately), we fail to find a systematic way to better utilize the power of query with larger k

Full solution

- Fortunately (or unfortunately), by building the decision tree with exhaustion, we can improve cost(5) from 14 to 12
- (In fact, 5 items can be sorted by 3 queries with $k = 4$)
- By using Merge Insertion Sort with the previously mentioned optimization, we can achieve cost(100) = 1016
  (don't forget to consider cost$(5) = 12$ in the intermediate steps of dynamic programming)