Sorting Algorithms and Information Entropy

Zhao Qiyuan
Jiang Yanwei
Liu Xuekai

Shanghai Jiao Tong University

Information theory

2020.6.1

Outline

Outline

Some Practial Problems

Figure: Many types of drinks in supermarket

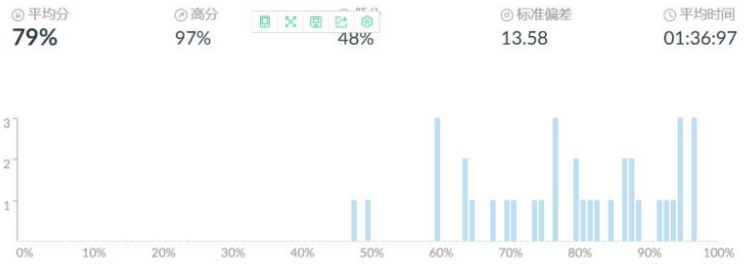

Figure: Our Mid-term grade of Signals and System

- How to sort these drinks or grades?

Some Typical Algorithms

Some naive algorithms:

- Insertion sort & Shell sort (Insert: $O(n^2)$
  Shell: approximately $O(n(logn)^2)$)
- Bubble sort $O(n^2)$
- Selection sort $O(n^2)$

Some superior algorithms

- Quick sort $O(nlogn)$
- Merge sort $O(nlogn)$
- Heap sort $O(nlogn)$

The red ones will be discussed emphatically.

Quick sort and heap sort

Quick sort

1. Pick an element, called a pivot, from the array.
2. Partitioning: elements less than pivot will be placed left than pivot, greater will be placed right.
3. Recursively apply the above steps to the sub-array.

Heap sort

Key point: Select the maximum element through a special data structure: heap
After building heap, we just need to fetch the maximum one and operate heap to keep the top is the maximum one.
Proof, analysis and improvemtent will be displayed later.

Outline

Our purpose

- Prove O(nlogn) is the lower bound of sorting algorithm.
- Improve Quick sort and Heap sort by adding or revising some details in inplementation.
- Prove and explain the second purpose by Information Theory.

Literature review

Any comparison algorithm requires $log(e(P))$ comparisons in the worst case, where $e(P)$ is the number of linear extensions of P. This is called the "information theory lower bound" (ITLB),and has been considered by Kislitsyn[1] in 1968. Almost 5 years later, Fredman[2] showed that sorting can be achieved with $log(e(P)) + 2n$ comparisons, where $n = |P|$.

Paul Hsieh[3] did a research on the running time of heapsort and quicksort in different compiling environments, which showed that the total CPU tally is different from the number of comparisons made.

Literature Review

Then David MacKey[4] thought the reason of that heapsort is not as fast as quicksort in general is it makes comparisons whose outcomes do not have equal prior probability.

Recently, a Chinese searcher called Qi Guo-qing[5] analyzed the efficiency of the insertion sort and merging sort by means of entropy, and pointed out that when the length of the series is larger than 4, any sorting algorithm based-on direct comparison requires more comparison than the lower bound.

But as for heapsort and quicksort, almost no one can give a reasonable and effective improvement method until now.

Outline

Lemma1

Lemma (lemma1)

$H(X^n) \leq \log(n!)$.*($H(X^n)$:result sequence)*

Proof.

The size of the sample space of X^n is $|\Omega| = n!$. Then it follows the conclusion in information theory straightforward. □

We will basically focus on the cases where X^n obeys a *uniform distribution*. That is, for any possible permutation p, $\Pr(X^n = p) = \frac{1}{n!}$.

Lemma2

Lemma (lemma2)

Let the information needed to sort the input sequence be Y. Then $H(X^n|Y) = 0$.

Proof.

Given Y, we can uniquely determine X^n. So $\exists! p, \Pr(X^n = p|Y) = 1$. Thus $H(X^n|Y) = 0$. □

Intuitively, in one comparison, the information that it contains is at most 1 bit, as the answer can be represented with 1 bit (0 for "no" and 0 for "yes"). So it seems reasonable to state that we need at least $\log_2(n!)$ comparisons. But actually, we can prove it in a formal way.

Lemma3

Lemma (lemma3)

Let Y be the information that we have obtained in previous comparisons. Then denote the answer of whether $a_i < a_j$ be $\mathrm{cmp}(a_i, a_j)$*, where* $1 \leq i \leq n, 1 \leq j \leq n$*. Then*

$$H(X^n|Y) - 1 \leq H(X^n|Y, \mathrm{cmp}(a_i, a_j)) \leq H(X^n|Y)$$

Combining the three lemmas above, we have the following theorem.

Proof of lemma3

Proof.

Since conditions reduce entropy,
$H(X^n|Y, \mathrm{cmp}(a_i, a_j)) \leq H(X^n|Y)$.
For another inequality, let $\Pr(a_i < a_j) = p$. Let the number of possible permutations under the condition Y be m. Then there are pm permutations where i is front of j and $(1-p)m$ permutations where i is behind j.
Since X^n obeys a uniform distribution, we have

$$\begin{aligned} H(X^n|Y, \mathrm{cmp}(a_i, a_j)) &= pH(X^n|Y, a_i < a_j) + (1-p)H(X^n|Y, a_i > a_j) \\ &= p\log_2(pm) + (1-p)\log_2[(1-p)m] \\ &= \log_2 m - H(p) \end{aligned}$$

Since $H(p) \leq 1$ and $H(X^n|Y) = \log_2 m$, we have
$H(X^n|Y) - 1 \leq H(X^n|Y, \mathrm{cmp}(a_i, a_j))$.

Theorem1

Theorem (theorem1)

(The information theoretic lower bound) The numbers of comparisons that are needed to sort a sequence with length n is at least $\log_2(n!)$.

Some textbooks also use the *decision tree model* to prove this lower bound. (Ref: Data Structure and Algorithm Analysis, 4ed) With this lower bound, we can prove the asymptotic time complexity of an optimal comparison-based sorting algorithm. It only requires some basic algebraic transformations.

Theorem2

Theorem (theorem2)

The asymptotic time complexity of an optimal comparison-based sorting algorithm is $\Omega(n \log n)$.

Proof.

$$\begin{aligned} \log_2(n!) &= \sum_{i=1}^{n} \log_2 i & &\geq \sum_{i=\lceil \frac{n}{2} \rceil}^{n} \log_2 i \\ &\geq \lceil \frac{n}{2} \rceil \log_2 \lceil \frac{n}{2} \rceil & &\geq \frac{n}{2} \log_2 \frac{n}{2} \\ &= \frac{n}{2} \log_2 n - \frac{n}{2} \end{aligned}$$

So $\log_2(n!) = \Omega(n \log n)$, when n is sufficiently large. □

Outline

Original Heapsort

A heap can be viewed as a binary tree. For a max(min)-heap, the *heap property* holds: for any element in the heap, it is larger(smaller) than its children. Thus the maximum(minimum) is at the root.

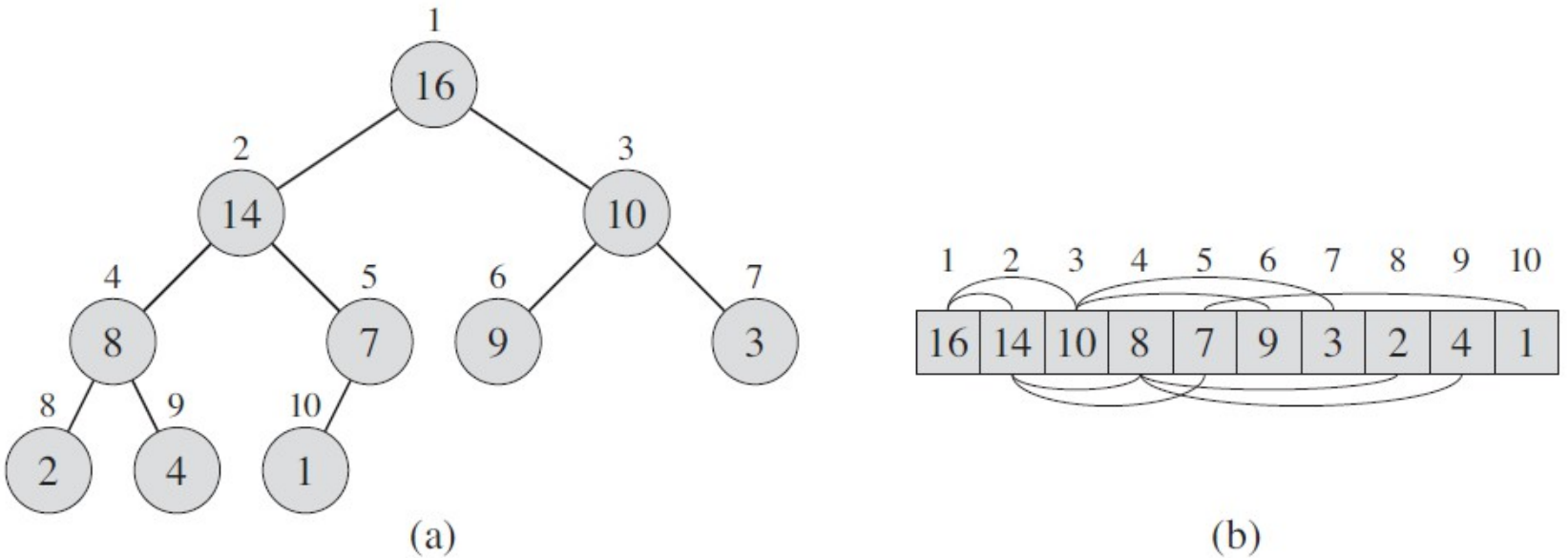

Figure: A max-heap viewed as (a) a binary tree and (b) an array. Picture credit: An Introduction to Algorithms, 3ed.

Fast Heapsort

Heapsort procedures:

1. Building the input sequence into a heap
2. Popping out the root of the heap
3. Maintaining the heap property

William used the largest index to replace the popped out one. The algorithm above can achieve the information theoretic lower bound asymptotically, but this swapping seems to disrupt the orderliness.

Explanation

So to make it more natural, MacKay used the leaves of root to replace root, and do that to the subtrees recursively, called 'fast heapsort'.

Although he did not make a more specific explanation for this method, we may understand it with the famous principle: *chaos increases entropy*. An similar example is that shuffling cards will increase the entropy of the order of cards, because it adds chaos into the order.

More improvement?

As we can see, in fast heapsort, the heap may not be an almost complete binary tree. To fix this, we can introduce a sifting-up operation (used in inserting an element into a heap) at the end of one iteration. This rearranged version is described below.

Input: a sequence A
Result: a sorted sequence A (in-place)

```
1 build_max_heap(A)
2 for i = len(A) downto 2 do
3 |   tmp := A[1]
4 |   remove A[1], resulting in a hole
5 |   pos := maintain(A, 1)
6 |   A[pos] := A[i]
7 |   sift_up(A, pos)
8 end
```

Algorithm 1: Fast Heapsort (Rearranged)

Testing

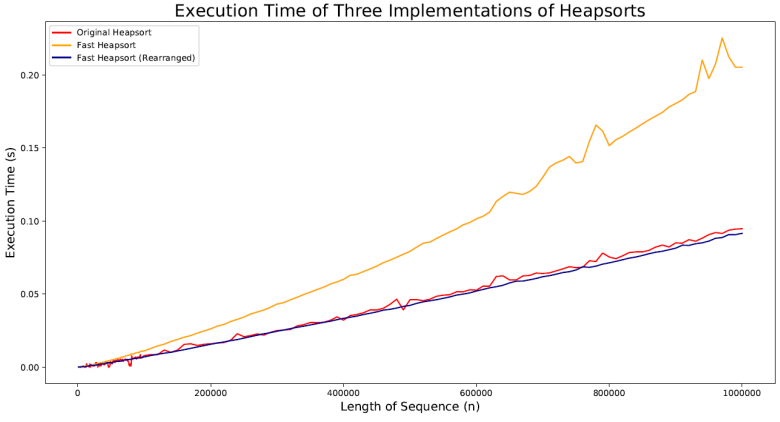

Figure: n is relatively small

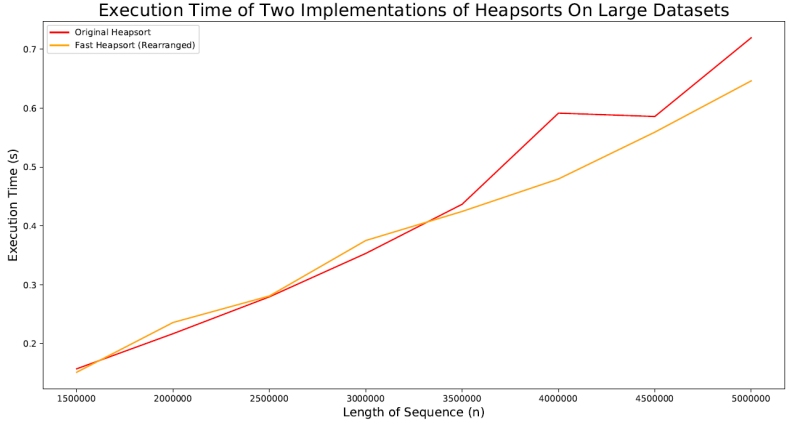

Figure: x is relatively large

DIscussion about the result

- Evaluation: time complexity and spatial complexity.
- Applicable conditions and stability
- Advantage and disadvantage

Conclusion of our work

- Sorting algorithm have a lower bound of time complexity: $\Omega(n \log n)$.
- Heapsort and quicksort can be improved with information theory.
- Experiments showed that improved version of heapsort is steadiest all the time, and fastest when n is larger than 100000.