A short note on probabilistic median algorithms

Yang Zonghan

July 4, 2020

1 Introduction

Sorting and selection problems are extensively researched by computer scientists and mathematicians. One of those is the median problem. The `Quick Select` algorithm provides a random algorithm with $O(n)$ time/space complexity, while there's a determined algorithm known as `Median of Medians` with $O(n)$ complexity as well. But if we run algorithm based on external data with comparatively small memory, things are much harder since $\Omega(\min\{m, n\})$ space is required for any selection algorithm with full correctness. So approximate streaming algorithm with good probabilistic guarantee is expected, and in this note I'll introduce some of them.

I come up with this topic when reviewing Algorithm finals. In 2019 fall Prof. Dominik Scheder taught basic streaming algorithms, while the time didn't permit us got into there this year. I was attracted by the median algorithm examined in 19' final, so I found materials online and learned a lot about this topic. This is mainly a short note on the topic, including associated materials and my thoughts of course.

2 Definition

Intuitively, median of a set of numbers is the number that at most half in the set are greater than it, and at most half in the set are less than it. It's quite straight forward, but unfortunately the informal definition — which works in many cases — fails in some cases.

Consider a very simple case : we have an array of numbers $\{1, 2, 3\}$. It's no doubt that 2 must be the median no matter how median defined. However, if we change 3 into 6 like we are playing dice with 6 faces, can we actually find a good, unique median? In fact, any number in $[3, 4]$ can be the median by the informal definition, which may disappoint us if we want the exact and unique median. Some suggest that 3.5 could be a proper median, some other against it object it by $3.5 \notin [6]$.

So in the median problem we have some restriction on input data, and the problem formally stated as follows:

> Given an array $\mathcal{A}$ of n distinct numbers from universe $\mathcal{U}$, where $|\mathcal{U}| = m, n = 2k + 1 (k \in \mathbb{Z})$.
> Find the number ranked $k + 1$ in the array.

Definition 2.1. $\mathrm{rank}(x) = \left|\{y \in \mathcal{A} \mid y \leq x\}\right|$

3 $O(n)$ median algorithms

$O(n)$ algorithms are fundamental to streaming algorithms. Firstly, the well-known `Quick Select` algorithm is described as follows:

Algorithm 1 Select the rank k element from a list A

```
procedure QuickSelect(X, k)
    if |X| = 1 then
        return X[1]
    else:
        p := uniformly random choice of element in X
        Y := [x ∈ X | x < p]
        Z := [x ∈ X | x > p]
        if |Y| = k − 1 then
            return p
        else if |Y| ≥ k then
            return QuickSelect(Y, k)
        else
            return QuickSelect(Z, k − |Y| − 1)
        end if
    end if
end procedure
```

It's $O(n)$ considering expected time and space and can be implemented in-place. How can we announce the algorithm is $O(n)$? Consider the fact that : the scale of data in the recursion will shrink to $\frac{3}{4}$ of former data w.p. at least $\frac{1}{2}$. This clearly shows an upper bound of expected complexity $O(n)$. If we sum up the expectation more carefully, the constant of `Quick Select` is quite small — generally the expected time is $T(n) = 2n + o(n)$, thus runs fast in practice.

There is also a determined algorithm for selecting numbers called `Median Of Medians`. It's the same with `Quick Select` except for finding pivot: it divides array into K parts, find approximate medians recursively, and then report the median of approximate medians as the pivot. Choosing $K = 5$, the data will shrink at least $\frac{7}{10}$ when recursing. It's slow since the constant in the worst case is greater than 30, but it provides a determined algorithm anyway, making `Quick Select` more reasonable.

4 Streaming Algorithm

Imagine you are a programmer in the 1970s who want to maintain a database recorded on the 200kb tape — the state-of-the-art in storage technology, with a 64kb memory computer. It's impossible to store all information in memory, but you also want the statistical result of the whole dataset, and you can only access data by reading it as a sequence of items by limited direction. Suppose you want to find the minimum and maximum of the data. It's quite easy — all you need to keep is the current optimum of data passed. However, if you want to find a median of the data, things are complicated since there's a theorem said that [1]:

Theorem 4.1. *In one pass, selection requires $\Omega(\min\{m, n\})$ space, where n is the size of input data and m is the universe of input data.*

It's awful bad news for you. However, streaming algorithms which are aimed at processing data streams in only a few passes (and sometimes one — if you replace tape with network when dealing with router tasks) with limited storage can help you with probabilistic and approximate result. That is to say, run at most p passes, report the ε-approximate answer with δ probability fail. Sometimes ε, δ

can be zero, so it may always report accurate answer when succeeded, or always successfully report an approximate answer.

Let's take a Morris' Algorithm as an example which is simply a counter of things [2]. A genuine counter who can count n numbers requires $\log n$ space. However, Morris improved it using probabilistic method by $O(\log \log n)$, which will always output an approximate answer.

Algorithm 2 Morris' algorithm with single counter

```
procedure Morris(S : data stream)
    X := 0
    while S.consume_next() do
        if w.p. 2^{-X} then
            X := X + 1
        end if
    end while
    return 2^X − 1
end procedure
```

Let's calculate the expected output. Let X_n be X after consuming n elements.

$$\begin{aligned}
\mathbb{E}[2^{X_n}] &= \sum_{j=0}^{\infty} \Pr[X_{n-1} = j]\mathbb{E}[2^{X_n} | X_{n-1} = j] \\
&= \sum_{j=0}^{\infty} \Pr[X_{n-1} = j](\frac{1}{2^j} \cdot 2^{j+1} + (1 - \frac{1}{2^j}) \cdot 2^j) \\
&= \sum_{j=0}^{\infty} \Pr[X_{n-1} = j](2^j + 1) \\
&= 1 + \sum_{j=0}^{\infty} \Pr[X_{n-1} = j]2^j \\
&= 1 + \mathbb{E}[2^{X_{n-1}}]
\end{aligned}$$

Since $\mathbb{E}[2^{X_0}] = 2^0 = 1$,

$$\mathbb{E}[2^{X_n}] = n + 1$$

But expectation isn't everything. Consider the variance of the result by calculating $\mathbb{E}[2^{2X_n}]$.

$$\begin{aligned}
\mathbb{E}[2^{2X_n}] &= \sum_{j=0}^{\infty} \Pr[2^{X_{n-1}} = j]\mathbb{E}[2^{2X_n} | 2^{X_{n-1}} = j] \\
&= \sum_{j=0}^{\infty} \Pr[2^{X_{n-1}} = j](\frac{1}{j} \cdot 4j^2 + (1 - \frac{1}{j}) \cdot j^2) \\
&= \sum_{j=0}^{\infty} \Pr[2^{X_{n-1}} = j](j^2 + 3j) \\
&= \mathbb{E}[2^{2^{X_{n-1}}}] + 3\mathbb{E}[2^{X_{n-1}}]
\end{aligned}$$

Taking the fact that $\mathbb{E}[2^{2X_0}] = 1$, it's easy to verify that

$$\mathbb{E}[2^{2X_n}] = \frac{3}{2}n^2 + \frac{3}{2}n + 1$$

Thus

$$\mathrm{Var}[2^{X_n}] = \mathbb{E}[2^{2X_n}] - \mathbb{E}^2[2^{X_n}] = \frac{1}{2}n^2 - \frac{1}{2}n = O(n^2)$$

It's actually a bad result: the output is too divergent. Figure 1 is an experiment testing $n = 2^{16} - 1$ for 10^6 cases. Also, notice that for the case when $n = 3 \cdot 2^k (k \in \mathbb{N})$ the approximate answer differs from accurate result at least $\frac{1}{3}$.

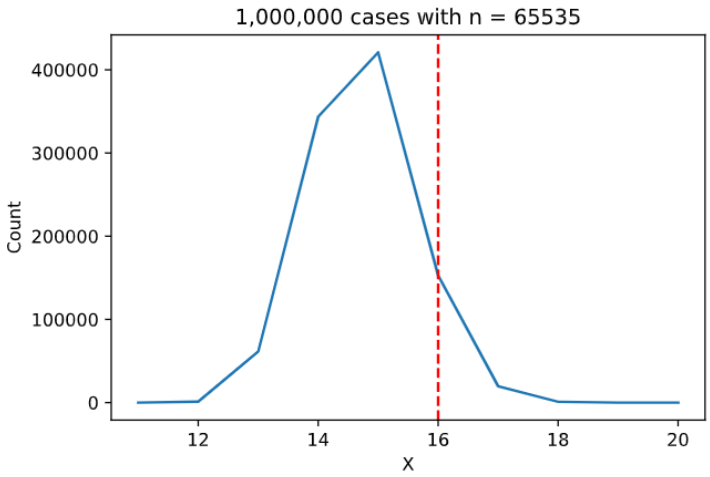

Figure 1: Running Morris using a single counter, $n = 2^{16} - 1, T = 10^6$.

Can we fix the issue? A simple idea is to take the average of t counters, and its correctness can be easily verified by Chebyshev inequality.

Theorem 4.2. *Let* $Y = \frac{1}{t}(\sum_{i=1}^{t} 2^{X_n^{(i)}} - 1)$. *The followings are held:*

- $\mathbb{E}[Y] = n$
- $\mathrm{Var}[Y] = \frac{1}{t}\mathrm{Var}[X_n]$
- *For* $t = \frac{1}{\varepsilon^2}$, $\Pr\left[Y \in [n - \varepsilon n, n + \varepsilon n]\right] > \frac{1}{2}$

Proof. Only the last claim is non-trivial. By Chebyshev's inequality,

$$\begin{aligned}\Pr[|Y - n| > \varepsilon n] &\leq \frac{1}{\varepsilon^2 n^2}\mathrm{Var}[Y] \\ &= \frac{1}{\varepsilon^2 n^2}\varepsilon^2(\frac{1}{2}n^2 - \frac{1}{2}n) \\ &= \frac{1}{2} - o(1)\end{aligned}$$

Therefore $\Pr\left[Y \in [n - \varepsilon n, n + \varepsilon n]\right] = \frac{1}{2} + o(1)$. □

The algorithm can be improved to δ failure and technique used is similar to the median problem we'll show later.

Theorem 4.3. *Repeating the* t*-counter sketch* $r = \lceil -\ln \delta \rceil$ *times results* $Z_1, \cdots, Z_r$. *Let the median of* $\{Z_1, \cdots, Z_r\}$ *be* $Z^{(m)}$. *Then*

$$\Pr\left[|Z^{(m)} - n| > \varepsilon n\right] < \delta$$

5 Streaming Median Algorithms

Let's get back to the topic we mainly focus on. Here I quote the streaming median problem statement by Dominik.

> In the streaming setting, your algorithm has one-time access to the data stream; it cannot rewind or access past elements. Its memory is bounded by $O(\log n + \log m)$. Our goal is to compute an approximate median of $\{x_1, \cdots, x_n\}$.
>
> 1. Design a streaming algorithm that, using randomness, outputs an element X such that $\mathbb{E}[\mathrm{rank}(X)] = \frac{m+1}{2}$.
> 2. Design a streaming algorithm that, using randomness, outputs an element X such that $\Pr[\mathrm{rank}(X) \in [0.49m, 0.51m]] > 0.99$. Prove that your algorithm works.

The first problem is quite simple — one possible solution is to randomly pick an element and simply output it. The correctness is straightforward : $(\sum_{i=1}^{n} i)/n = (n+1)/2 = k+1$. Now one problem is that we don't have random access of the data, so sampling is something we need to concern — though it can be solved easily using $t + O(1)$ registers by the following algorithm, where t is the sample size.

Algorithm 3 Sample k elements from data stream

```
1:  procedure Sample(S : data stream, t : size of sample)
2:      sample := ∅
3:      n := 0
4:      while S.has_next() do
5:          n := n + 1
6:          if n ≤ t then
7:              sample.append(S.get_next())
8:          else
9:              if w.p. t/n then
10:                 index := uniformly random choice of element in [k]
11:                 sample[index] := S.get_next()
12:             else
13:                 S.pop_next()
14:             end if
15:         end if
16:     end while
17:     return sample
18: end procedure
```

A simple induction on i can verify the claim: $\forall i \in \{t, \cdots, n\}$, the sample kept when $n = i$ is uniformly chosen from all possible k samples of first i elements. So the correctness of the algorithm is granted.

Good expectation may mean nothing. If we simply output the randomly chosen element, it helps us to understand the streaming data very little. To bound the accuracy of output, we shall keep a larger sample, and conclude a good result from the sample. However, the method — mean — we use in the last task can lead to some problem. Consider the stream consisting of $\frac{2}{3}n$ items of 1 and $\frac{1}{3}n$ items of m. The median should be 1, but taking average of some 1 and some m will lead to a rank at least $\frac{2}{3}n$. Thus, it's very likely to have an output differs from answer at least $\frac{2}{3}n$.

A natural fix is to replace mean with median to match the problem statement. Surprisingly, this minor fix leads to a fairly good algorithm. Although theorems are mainly about means, there's a clever trick to solve the problem.

Algorithm 4 Report approximate median using one pass and sample size t

```
procedure ApprMedian(S : data stream, t : size of sample)
    sample := Sample(S, t)
    return QuickSelect(sample, (t + 1)/2)
end procedure
```

Theorem 5.1. *In the algorithm* `ApprMedian`*, if* $t = \frac{7}{\varepsilon^2} \ln \frac{2}{\delta}$ *for sufficiently small* $\varepsilon, \delta > 0$*, then the answer will be* ε*-median w.p.* $1 - \delta$*.*

Proof. Firstly we invoke Chernoff bound, a corollary of Markov inequality.

Theorem 5.2. *(Chernoff bound) Let* $X_1, \cdots, X_t$ *be independent and identically distributed random variables with range* $[0, 1]$ *and expectation* μ*. For* $X = \frac{1}{t} \sum_i X_i$ *and* $\delta \in (0, 1)$*,*

$$\Pr[|X - \mu| > \delta\mu] < 2e^{-\mu t \delta^2/3}$$

Corollary. If $X_i \in [0, c]$*,*

$$\Pr[|X - \mu| > \delta\mu] < 2e^{-\mu t \delta^2/3c}$$

Partition the data in stream $\mathcal{S}$ into 3 groups

$$\begin{aligned} S_L &= \{x \in \mathcal{S} | \mathrm{rank}(x) < \frac{m}{2} - \varepsilon m\} \\ S_M &= \{x \in \mathcal{S} | \frac{m}{2} - \varepsilon m \le \mathrm{rank}(x) \le \frac{m}{2} + \varepsilon m\} \\ S_U &= \{x \in \mathcal{S} | \mathrm{rank}(x) > \frac{m}{2} + \varepsilon m\} \end{aligned}$$

Fact 5.3. *If less than* $\frac{t}{2}$ *elements from each of* S_L *and* S_U *are in sample then its median is in* S_M*.*

Let $X_i = 1$ if i-th sample is in S_L and 0 otherwise, $X = \sum X_i$. We need to calculate $\Pr[X \ge \frac{t}{2}]$. We omit the dependency of random variables, for some reasons:

- For those $t = o(n)$, the dependencies of random variables are too weak to consider.
- For those $t = O(n)$, the answer should be very exact and out of our estimation.
- It's actually scaling up the result, since with the growth of X the probability that X increases goes down.

By Chernoff,

$$\begin{aligned} \Pr[X \ge \frac{t}{2}] &\le \Pr[X \ge (1 + \varepsilon)\mathbb{E}[X]] \\ &\le e^{-\frac{\varepsilon^2(1/2 - \varepsilon)t}{3}} \\ &= (\frac{2}{\delta})^{-\frac{7}{3}(\frac{1}{2} - \varepsilon)} < \frac{\delta}{2} \end{aligned}$$

Let $Y_i = 1$ if i-th sample is in S_U and 0 otherwise, $Y = \sum Y_i$. An exactly the same argument on Y can be done as X, thus from fact 5.3, the failure rate can be expressed as

$$\Pr[X_i \geq \frac{t}{2} \wedge Y_i \geq \frac{t}{2}] \leq \Pr[X_i \geq \frac{t}{2}] + \Pr[Y_i \geq \frac{t}{2}] < \delta$$

□

The wonderful part of proof is the indicator random variable, with which theorems of means work for medians too. Remember theorem 4.3 we've stated before, the result is quite similar to theorem 5.1, and actually the proving method is, too, by partitioning sets and using Chernoff to study the indicator random variable. Only one thing is different: in Morris' algorithm the random variables are truly independently identically distributed.

In task 2, $\varepsilon, \delta = 0.01$, then $t \approx 3.7 \times 10^5$. It looks big, but it's $O(1)$ anyway.

Here's the result of some tests using the approximate median algorithm we've just described.

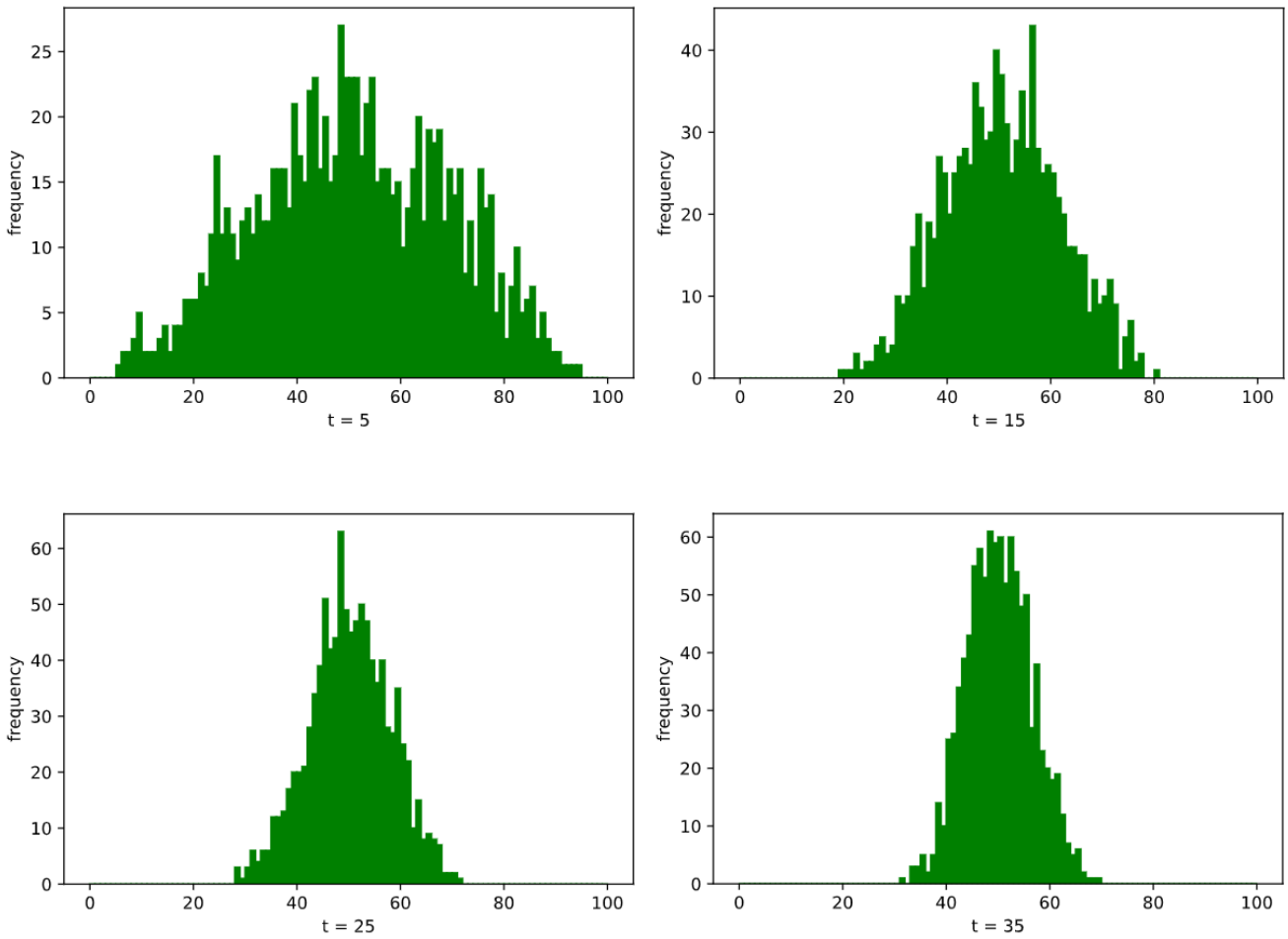

Figure 2: Running Approximate Median for 1000 cases, using different t with $n = 99$.

For some stream granted to be equally likely permuted, there's Munro-Paterson algorithm [3] which reports the exact median in one pass with δ failure rate, shown in Algorithm 5.

Algorithm 5 Report exact median using one pass and memory of $r + O(1)$ registers

```
1: procedure MunroPaterson(S : data stream, r : size of memory)
2:     s := ∅
3:     n, l, h := 0
4:     while S.has_next() do
5:         n := n + 1
6:         x := S.get_next()
7:         if x < s.min() then l := l + 1
8:         else if x > s.max() then h := h + 1
9:         else
10:            s.add(x)
11:            if s.size() = r + 1 then
12:                if h < l then
13:                    s.pop_max()
14:                    h := h + 1
15:                else
16:                    s.pop_min()
17:                    l := l + 1
18:                end if
19:            end if
20:        end if
21:    end while
22:    if 1 ≤ (n + 1)/2 − l ≤ r then
23:        return s.get((n + 1)/2 − l)
24:    else
25:        return FAIL
26:    end if
27: end procedure
```

To analyze the algorithm we consider the random variable $d = h - l$ which starts at 0. The algorithm fails only if $|d|$ at the end of the stream is greater than r. Let's examine the change of d.

For the first r iterations, s filled with first r items of $\mathcal{S}$ with no change of h, l. Consider the iteration i when $i > r$. The probability that $\mathcal{S}_i$ will be larger that max s is $(h+1)/(h+r+l+1)$, and the probability that $\mathcal{S}_i$ will be smaller that min s is $(l+1)/(h+r+l+1)$.

Let $p_{d,i}$ be the probability that $|d|$ increases by 1 conditioned on $0 \leq |d| < r$. If $d >= 0$,

$$p_{d,i} = \frac{h+1}{h+r+l+1} \leq \frac{h+1}{h+(h+1)+1} = \frac{1}{2}$$

If $d < 0$,

$$p_{d,i} = \frac{l+1}{l+r+h+1} \leq \frac{l+1}{l+(l+1)+1} = \frac{1}{2}$$

Therefore,

$$p_{d,i} \leq \frac{1}{2}$$

Loosen the condition we can get a sufficient condition for algorithm's success: $|d| < r$ for all time. Since $p_{d,i} \leq \frac{1}{2}$, we can scale up the problem to the escape time of a random walk — it's actually a random walk with 'centripetal force', and losing the 'centripetal force' only makes the escape time earlier. A theorem in Feller's book said that [4]:

Theorem 5.4. *(Limit theorem for first passages) For fixed t the probability that the first passage through r occurs before tr^2 tends to*

$$\sqrt{\frac{2}{\pi}} \int_{1/\sqrt{t}}^{\infty} e^{-\frac{1}{2}s^2} ds = 2\left[1 - \mathfrak{N}(\frac{1}{\sqrt{t}})\right]$$

as $r \to \infty$, where $\mathfrak{N}$ is the normal distribution function.

Toughly speaking, the waiting time for the first passage through r increases with the square of r. It follows immediately that for any $\delta > 0$, there's a suitable t such that with memory sized $r = t\sqrt{n}$, the failure rate is at most δ. Therefore,

Theorem 5.5. *For a single-pass algorithm which nearly always finds the median facing equally likely permuted input, $\Omega(\sqrt{n})$ locations are sufficient.*

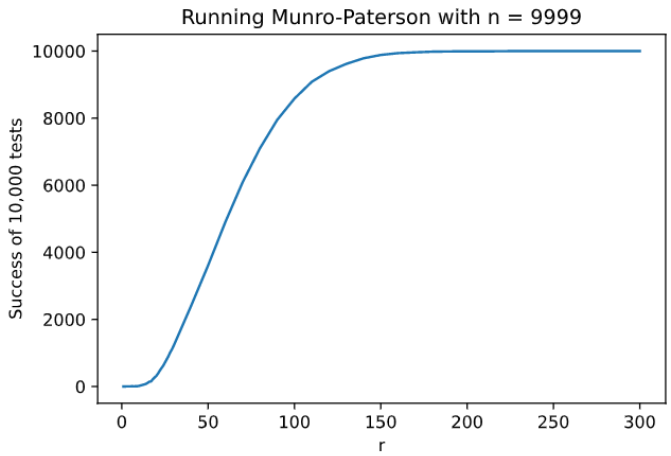

Figure 3: Running Munro-Paterson using different r with $n = 9999$.

Let's run some tests to show the accuracy. For $n = 9999$, the success rate of different r is shown in figure 3. It's quite impressive that with $t = 2$ the failure rate is less than 10^{-4}.

Munro-Paterson algorithm can be extended to a multi-pass algorithm, which almost success with p passes using $\Omega(n^{1/p})$ space. Also, the author of the algorithm argued that with some distribution the minimum space required for one-pass algorithm is $\Theta(\sqrt{n})$. Therefore, Munro-Paterson algorithm is a kind of 'best' algorithm for equally likely permuted input.

6 Summary

This note introduced many algorithms about finding the median, and most of them are taking advantage of randomness. `Quick Select` shows efficiency with simple summing of the expectation. `Approximate Median`'s correctness is bounded by Chernoff bound by using indicator random variable to transform median into mean. `Munro Paterson` is an excellent algorithm facing equally likely distributed input data, and we show the correctness by scaling up the analysis to a 1-d random walk problem. All the proofs are quite impressive and elegant.

There's a note by Steven J. Miller [5] introduced the median variation of many traditional theorems by methods of order statistics, including Law of Large Numbers and Central Limit Theorem. It may be too far from our main topic, but I strongly suggest having a read about it since it's quite interesting.

Acknowledgments

I'd like to give my sincere gratitude to Prof. Wu for the wonderful semester of probability theory.

I'd also like to thank Prof. Dominik Scheder for the wonderful algorithm course and the inspiration of topic of this note.

I have to thank particularly Mao Xinyu I learned a lot about (but not limit to) this topic and lead me to materials that truly helped me.

Also, I really appreciate Zhuang Yonghao, Zhang Yuheng for finding bugs in my note.

The course $o(n)$ *Sublinear Algorithms for Big Datasets* by Prof. Grigory Yaroslavtsev [6] helps me understand the topic a lot, however I didn't find a chance to mention it at the main part.

References

[1] David Dobkin and J. Munro. Time and space bounds for selection problems. pages 192–204, 07 1978.

[2] Robert Morris. Counting large numbers of events in small registers. *Commun. ACM*, 21(10):840–842, October 1978.

[3] J. Ian Munro and Mike Paterson. Selection and sorting with limited storage. *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, pages 253–258, 1978.

[4] Willliam Feller. *An introduction to probability theory and its applications, vol 1.* John Wiley & Sons, 2008.

[5] Steven J. Miller. *The Probability Lifesaver:Order Statistics and the Median Theorem*, 2015.

[6] Grigory Yaroslavtsev. *o(n) Sublinear Algorithms for Big Datasets*, 2014.