size.R

Preston

2020-05-23

```
# Preston Dunton
# CS320 Honors Option
# May 20, 2020
# pdunton@rams.colostate.edu

# size() can be implemented in an O(1) and an O(logn) way.
# To do it in O(1) time, keep a size_ member in the Binomial_Heap object
# and update it for insertions / deletions.  In this case, size() just returns that field.
# To do it in O(logn) time, iterate through the linked list of binomial trees
# and use the degree of the head of each tree to calculate how many nodes are in each tree.
# Because the length of that list is at most logn, size() is O(logn).
# I implemented the O(logn) way because I didn't feel like writing the O(1) implementation.

size_binomial = read.csv("./size_binomial.csv")
attach(size_binomial)
```

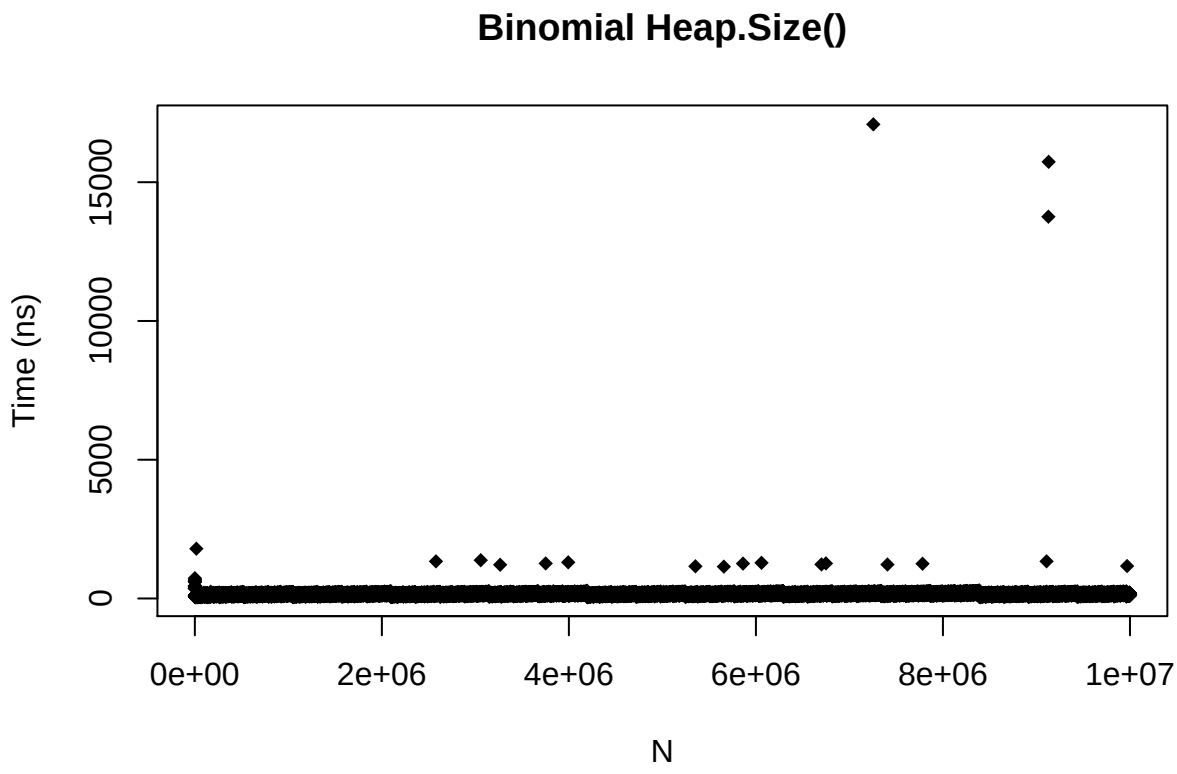
```
## The following object is masked from package:base:
##
##     T
```

```
summary(T)
```

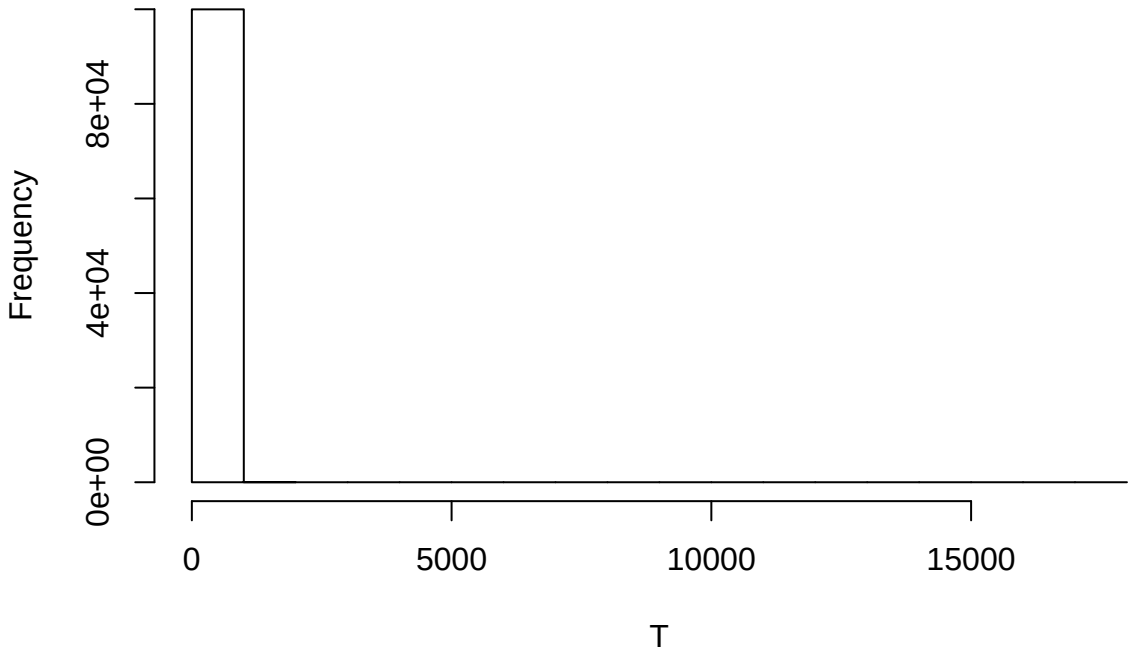
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    45.0   140.0   157.0   160.6   181.0 17085.0
```

```
# min 45
# q1 140
# median 157
# mean 160
# q3 181
# max 17085

plot(N,T,pch=18,xlab="N",ylab="Time (ns)",main="Binomial Heap.Size()")
```




```
hist(T,breaks=20)
```




```
# Let's see if we can remove some outliers

quantile(T,seq(0,1,0.1))
```

```
##     0%    10%    20%    30%    40%    50%    60%    70%    80%    90%   100%
##     45    124    137    142    153    157    167    179    183    196 17085
```

```
quantile(T,seq(0.9,1,0.01))
```

```
##    90%    91%    92%    93%    94%    95%    96%    97%    98%    99%   100%
##    196    198    205    206    207    208    210    219    221    232 17085
```

```
# Let's separate the top 1% and analyze

# Top 1%
  sum(T>232) # There are 986 outliers
```

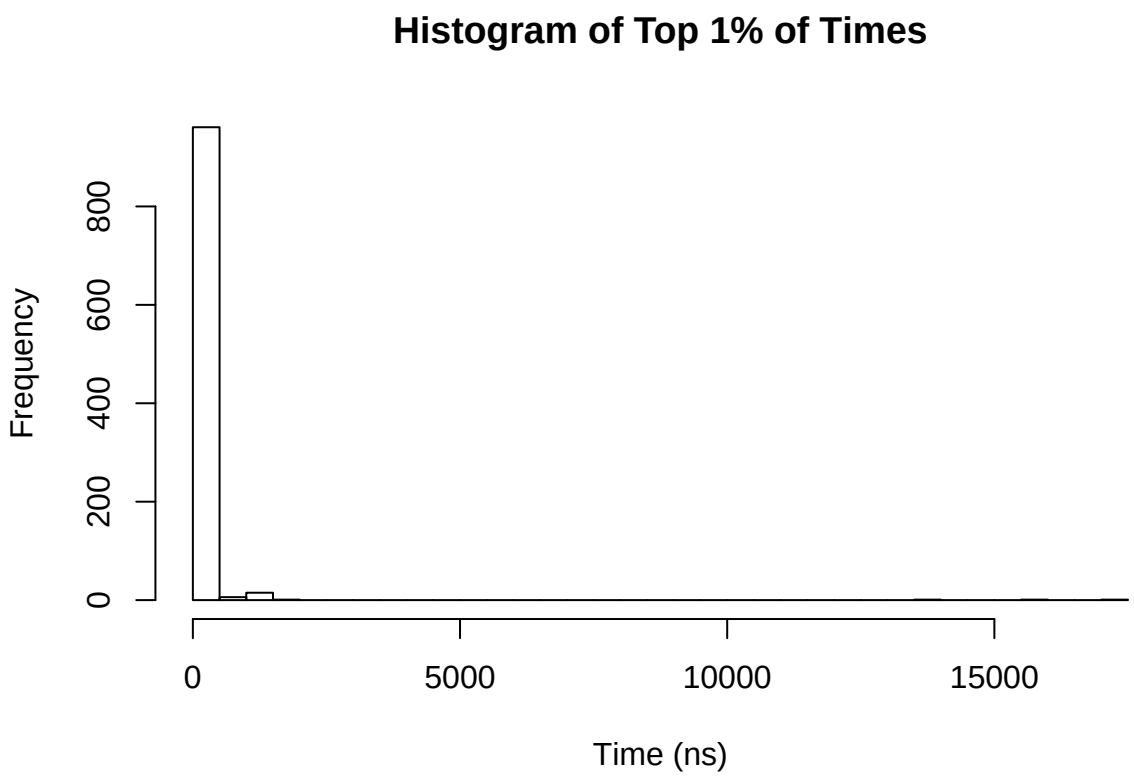
```
## [1] 986
```

```
  summary(T[which(T>232)])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   233.0   234.0   236.0   307.4   246.0 17085.0
```

```
  # min 233
  # q1 234
  # median 236
  # mean 307.4
  # q3 246
```

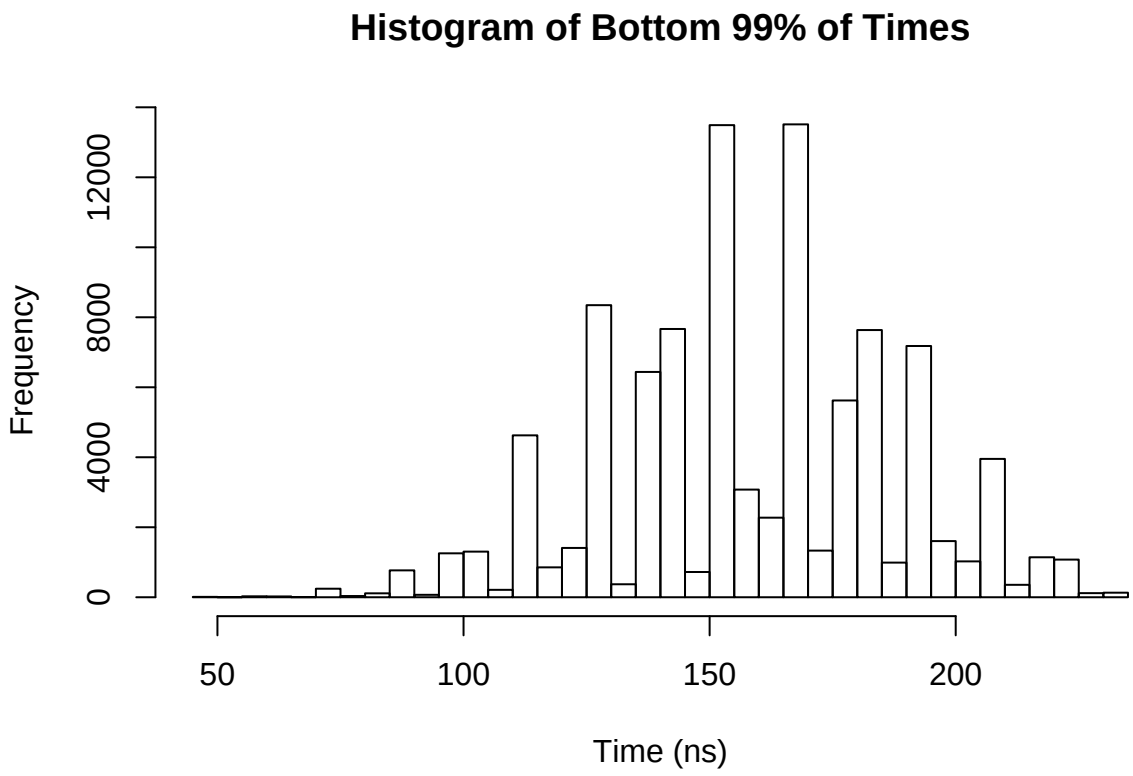
```r
  # max 17085
  hist(T[which(T>232)],main="Histogram of Top 1% of Times",xlab="Time (ns)",breaks=30)
```




```r
# Bottom 99%
  summary(T[which(T<=232)])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    45.0   140.0   156.0   159.1   181.0   232.0
```

```r
  # min 45
  # q1 140
  # median 156
  # mean 159.1
  # q3 181
  # max 232
  hist(T[which(T<=232)],main="Histogram of Bottom 99% of Times",xlab="Time (ns)",breaks=30)
```




```
  # Looks like most calls to size() take under 232 ns.
  # There also doesn't appear to be a large correlation between T and N
  # The implementation appears to be O(logn) time like the other operations that have this trend.
  # An O(1) implementation could be achieved by implementing a size_ member, but I don't want to
  # bother with that.

detach(size_binomial)
```