Assignment 3 - Running Time Analysis of 7 Functions

Hieu Duong

July 8, 2022

Contents

Cite: Analysis of Algorithms - Deriving Cost Function (codesdope.com)

Assuming worst case is when the operation count of a function is highest given inputs of the same size but of different organization.

Question 1

1. This function's worst, best, average cases are the same. Its cost function is:

$$t_{(\text{cartesianProduct})} = 4n^2 + 5n + 2$$

   Analysis:

   a. The outer loop runs n times, performing a number of operations in each iteration (call it c_1). The function performs 2 operations outside that loop, namely the initialization of i and the terminating loop condition check. Thus, its cost function is: $C = c_1 \cdot n + 2$

   b. For each iteration of the outer loop, the inner loop runs n times, performing a number of operations in each iteration (call it c_2). The outer loop's body performs 5 operations outside that inner loop. Thus, $c_1 = c_2 \cdot n + 5$

   c. Each iteration of the inner loop has 4 operations. Thus, $c_2 = 4$

   d. Plugging c_2 into c_1, and then c_1 into C gives $t_{(\text{cartesianProduct})}$ above:
   $C = (4n + 5) \cdot n + 2 = 4n^2 + 5n + 2$

2. The function's barometer operations are:

   a. The inner *while* loop comparison: `(j < n)`
   b. The printing of Cartesian products to standard output: `cout << "{" << arr[i] << "," << arr[j] << "}";`
   c. The increment of the inner loop control variable j: `j++;`
   d. The printing of white spaces to standard output: `cout << " ";`

3. The function's O notation running time is $O(n^2)$.

Question 2

1. This function's worst, best, average cases are the same. Its cost function is:

$$t_{(\text{triangle})} = 3n^2 + 13n + 3$$

   Analysis:

   a. The function has two outer loops, one after the other, with the operation count in each of their iteration being c_1 and c_2. The first outer loop runs n times, the second one runs an undetermined number of times (call it m). In addition to these 2 loops, the function also performs 3 other operations, namely the initialization of i and the 2 terminating loop condition checks. Its cost function is therefore: $C = c_1 \cdot n + c_2 \cdot m + 3$

   b. For each iteration of the first outer loop, its inner loop runs an undetermined number of times (call it o) with each of its iteration having 3 operations. Besides this inner loop, the first outer loop's body also performs 5 other operations. Thus, $c_1 = 3o + 5$

      i. Tracing through this loop, it's apparent that:
      $i = 0 \rightarrow o = 1$
      $i = 1 \rightarrow o = 2$
      ...
      $i = n - 1 \rightarrow o = n$
      $i = n \rightarrow o = n + 1$

      ii. Expressing o as a function of i gives: $o(i) = i + 1$

iii. The cost function of the first outer loop thus becomes:
$c_1(i) = 3(i+1) + 5 = 3i + 8$

iv. As it's obvious c_1 is not a constant, the cost function of the main function needs to be modified to reflect that:
$C = \sum_{i=0}^{n-1} c_1(i) + c_2 \cdot m + 3$

v. Evaluating the first term only gives:
$\sum_{i=0}^{n-1}(3i+8) = 3 \cdot \sum_{i=0}^{n-1} i + \sum_{i=0}^{n-1} 8 = 3\frac{(n-1)n}{2} + 8n = \frac{3n^2}{2} + \frac{13n}{2}$

vi. Plugging this back into the main cost function gives:
$C = \frac{3n^2}{2} + \frac{13n}{2} + c_2 \cdot m + 3$

c. With the first outer loop ending after n iterations, the second outer loop thus begins with $i = n$. For each iteration of the first outer loop, its inner loop runs an undetermined number of times (call it p) with each of its iteration having 3 operations. Besides this inner loop, the second outer loop's body also performs 5 other operations. Thus, $c_2 = 3p + 5$

i. Tracing through this second inner loop, it's apparent that:
$i = n \rightarrow p = n$
$i = n-1 \rightarrow p = n-1$
$\ldots$
$i = 1 \rightarrow p = 1$
$i = 0 \rightarrow p = 0$

ii. Expressing p as a function of i gives: $p(i) = i$

iii. The cost function of the second outer loop thus becomes:
$c_2(i) = 3 \cdot p(i) + 5 = 3i + 5$

iv. As the second outer loop runs between $i = n$ and $i = 1$ inclusive, its cost function is thus:
$\sum_{i=1}^{n} c_2(i) = \sum_{i=1}^{n}(3i+5) = \frac{3n^2}{2} + \frac{13n}{2}$

v. Plugging this back into the main cost function gives $t_{(\text{triangle})}$ above:
$C = \left(\frac{3n^2}{2} + \frac{13n}{2}\right) + \left(\frac{3n^2}{2} + \frac{13n}{2}\right) + 3 = 3n^2 + 13n + 3$

2. The function's barometer operations are:

   a. The inner *while* loops comparison: `(j <= i)`
   b. The printing of numbers and whitespaces to standard output: `cout << j << " ";`
   c. The increment of the inner loops control variable j: `j++;`

3. The function's O notation running time is $O(n^2)$.

Question 3

1. This function's worst, best, average cases are the same. Its cost function is:

$$t_{(\text{matrixSelfMultiply})} = 5n^3 + 7n^2 + 4n + 4$$

Analysis:

a. The function has 1 outermost loop that runs n times from $r = 0$ to $r = n-1$ inclusive, with each of its iteration costing $c_1(r)$. Apart from this loop, the function also performs 4 other operations. The total cost function is thus:
$C = 4 + \sum_{r=0}^{n-1} c_1(r)$

b. Each iteration of this loop has another loop inside and 4 other operations. This inner (middle) loop also runs n times from $c = 0$ to $c = n-1$ inclusive, with each of its iteration costing $c_2(c)$. The cost function of this middle loop is thus:
$c_1(r) = 4 + \sum_{c=0}^{n-1} c_2(c)$

c. Each iteration of this middle loop has another loop inside and 7 other operations. This innermost loop also runs n times from iNext $= 0$ to iNext $=$ n $- 1$ inclusive, with each of its iteration costing 5 operations. The cost function of this middle loop is thus:
   $c_2(c) = 7 + \sum_{\text{iNext}=0}^{n-1} 5 = 5n + 7$

   i. Plugging that back into the outermost loop's cost function gives:
      $c_1(r) = 4 + \sum_{c=0}^{n-1}(5n + 7) = 4 + n(5n + 7) = 5n^2 + 7n + 4$

   ii. Plugging that back into the main cost function gives $t_{(\text{matrixSelfMultiply})}$ above:
      $C = 4 + \sum_{r=0}^{n-1}(5n^2 + 7n + 4) = 4 + n(5n^2 + 7n + 4) = 5n^3 + 7n^2 + 4n + 4$

2. The function's barometer operations are:

   a. The innermost *while* loop comparison: `(iNext < rows)`
   b. The computation of the variable: `next += m[rcIndex(r, iNext, columns)] * m[rcIndex(iNext, c, columns)];`
   c. The increment of the innermost loop control variable: `iNext++;`

3. The function's O notation running time is $O(n^3)$.

Question 4

1. This function's worst case is with reverse sorted arrays. Its cost function then is:

$$t_{(\text{ssort})} = \begin{cases} 1 & , \quad n = 0 \\ \left\lfloor \frac{7}{4}n^2 + \frac{11}{2}n - 6 \right\rfloor & , \quad n > 0 \end{cases}$$

   Analysis:

   a. This function recurses $n - 1$ times from $i = 0$ to $i = n - 2$ inclusive, with each of its execution costing $c_1(i)$. Together with the terminating condition check, the total cost function is thus:
      $C = 1 + \sum_{i=0}^{n-2} c_1(i)$
   b. In all cases, the first *if* statement will evaluate to true in each execution but the last, and costs 1 operation each. When the *if* body is executed, there's a loop inside and 6 other operations. That loop runs from next $= i+1$ to next $= n-1$ inclusive, for a total of $(n-1)-(i+1)+1 = n-i-1$ times, with each of its iteration costing $c_2(\text{next})$. The cost of each recursive execution but the last is thus:
      $c_1(i) = 7 + \sum_{\text{next}=i+1}^{n-1} c_2(\text{next})$
   c. Each iteration of this loop contains 2 operations and an *if* statement. That *if* condition is always checked in each loop iteration, and each time this *if* evaluates to true, there's 1 operation inside its body. In the function's worst case, this *if* body will be executed some of the time (?).

      - I gave up and used statistical regression instead.

2. The function's barometer operations are:

   a. The *while* loop comparison: `(next < n)`
   b. The inner *if* statement comparison: `(arr[next] < arr[smallest])`
   c. The increment of the *while* loop control variable: `next++;`

3. The function's O notation running time is $O(n^2)$.

Question 5

1. This function's worst, best, average cases are the same. Its cost function is:

$$t_{(\text{pattern})} = 3n\log_2 n + 23n - 9$$

   Analyzed with statistical regression.

2. The function's barometer operations are:

   a. The *while* loop comparison: `(ast < n)`
   b. The printing of asterisks to standard output: `cout << "* ";`
   c. The increment of the *while* loop control variable: `ast++;`

3. The function's O notation running time is $O(n \log n)$.

Question 6

1. This function's worst case is with the desired element at the last array index. Its cost function then is:

$$t_{(\text{lsearch})} = \begin{cases} 1 & , \quad n = 0 \\ 3 \cdot 2^n - 4 & , \quad n > 0 \end{cases}$$

   Analyzed with statistical regression.

2. The function's barometer operations are:

   a. The first *if* statement comparison: `(len == 0)`
   b. The second *if* statement comparison: `(arr[0] == target)`

3. The function's O notation running time is $O(2^n)$.

Question 7

1. This function's worst case is with odd exponents. Its cost function then is:

$$t_{(\text{pow})} \leq 5\log_2(n+1) + 4$$

   Analyzed with statistical regression and manual fine-tuning.

   - Preliminary analysis suggests $5\log_2 n + 3$ and $4\log_2 n + 7$ as the algorithm's upper and lower bounds respectively. However, both of these fail to bound all data points when they get large or small.

2. The function's barometer operations are:

   a. The *while* loop comparison: `(exp > 0)`
   b. The *if* statement comparison: `(exp & 1)`
   c. The right shift bitwise computation: `exp >>= 1;`
   d. The squaring of the variable: `base = base * base;`

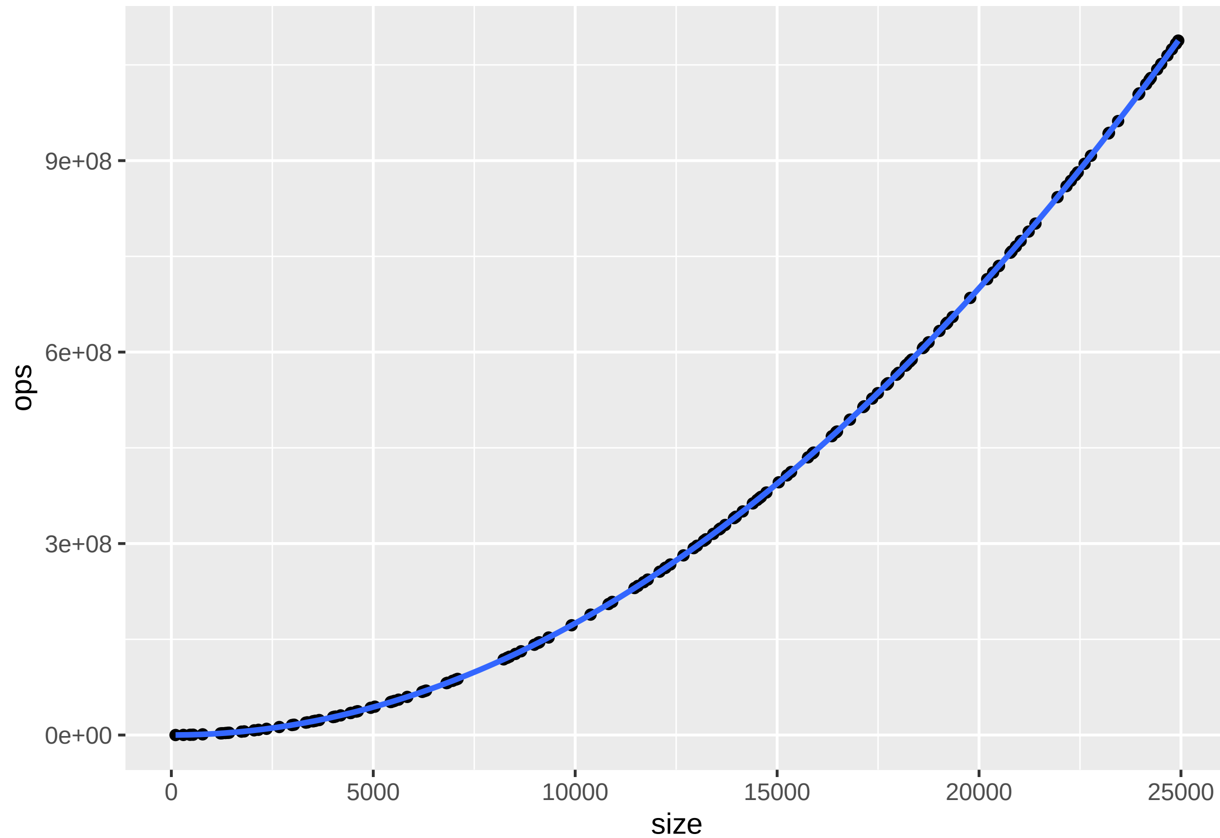
3. The function's O notation running time is $O(\log n)$.

Appendix A - Statistical Analyses

Minimum sample size = 30.

Question 4 - `ssort`

Make a scatterplot with a smooth line.




```
##       size            ops
##  Min.   :  104   Min.   :1.949e+04
##  1st Qu.: 5576   1st Qu.:5.446e+07
##  Median :13567   Median :3.222e+08
##  Mean   :12791   Mean   :3.830e+08
##  3rd Qu.:18693   3rd Qu.:6.116e+08
##  Max.   :24933   Max.   :1.088e+09
```

As it's obviously not linear, first try with a 10th degree polynomial.

```
##
## Call:
## lm(formula = ops ~ poly(size, 10), data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.21529 -0.12155  0.07355  0.10308  0.21430
```

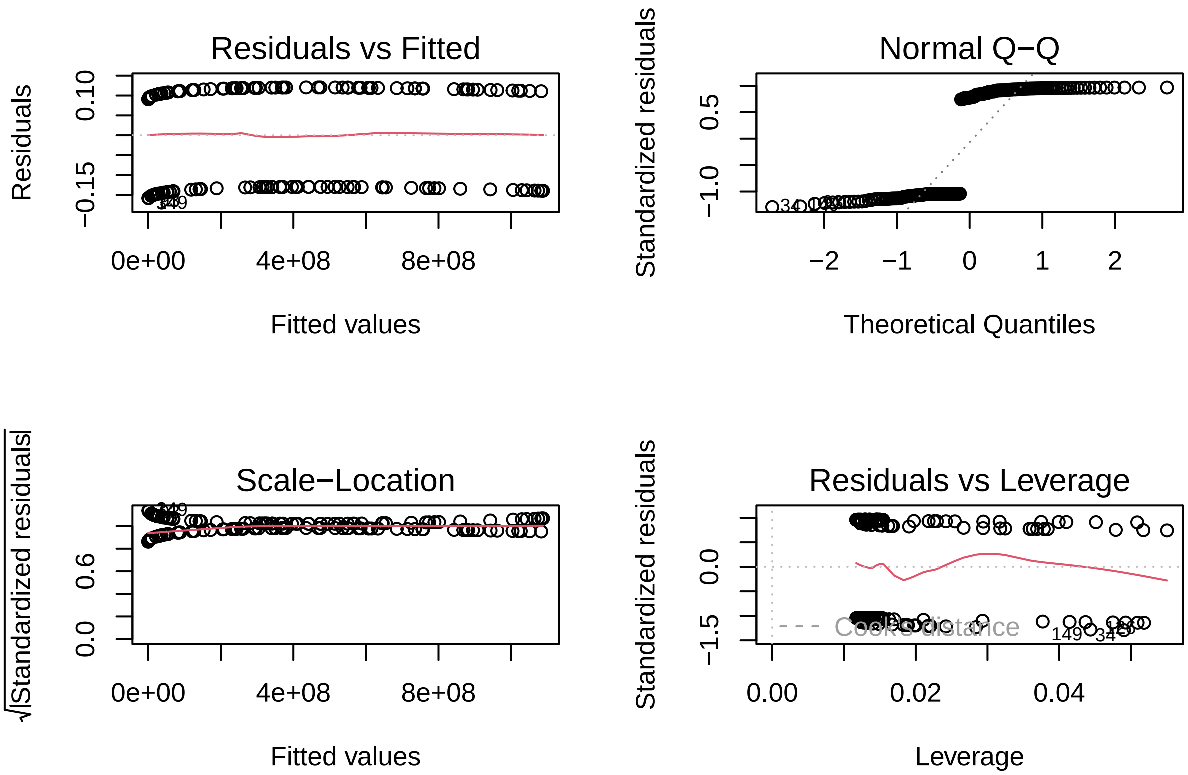
```
##
## Coefficients:
##                    Estimate Std. Error    t value Pr(>|t|)
## (Intercept)       3.830e+08  1.016e-02  3.771e+10   <2e-16 ***
## poly(size, 10)1   4.018e+09  1.248e-01  3.220e+10   <2e-16 ***
## poly(size, 10)2   1.029e+09  1.248e-01  8.243e+09   <2e-16 ***
## poly(size, 10)3  -9.168e-02  1.248e-01 -7.350e-01   0.4638
## poly(size, 10)4  -4.010e-03  1.248e-01 -3.200e-02   0.9744
## poly(size, 10)5  -2.991e-01  1.248e-01 -2.397e+00   0.0179 *
## poly(size, 10)6  -8.337e-02  1.248e-01 -6.680e-01   0.5052
## poly(size, 10)7   7.055e-02  1.248e-01  5.650e-01   0.5727
## poly(size, 10)8  -1.103e-01  1.248e-01 -8.840e-01   0.3781
## poly(size, 10)9  -6.357e-02  1.248e-01 -5.090e-01   0.6113
## poly(size, 10)10 -1.397e-01  1.248e-01 -1.119e+00   0.2650
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1248 on 140 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 1.105e+20 on 10 and 140 DF,  p-value: < 2.2e-16
```

Clear that a quadratic is all we need.

$$\mu_y = \beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon$$

```
##
## Call:
## lm(formula = ops ~ size + I(size^2), data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.15815 -0.13275  0.09531  0.11601  0.12034
##
## Coefficients:
##               Estimate Std. Error    t value Pr(>|t|)
## (Intercept) -6.090e+00  2.990e-02 -2.037e+02   <2e-16 ***
## size         5.500e+00  5.536e-06  9.935e+05   <2e-16 ***
## I(size^2)    1.750e+00  2.133e-10  8.206e+09   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1254 on 148 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 5.473e+20 on 2 and 148 DF,  p-value: < 2.2e-16
```

```
##                 2.5 %    97.5 %
## (Intercept) -6.149194 -6.031032
## size         5.499985  5.500007
## I(size^2)    1.750000  1.750000
```



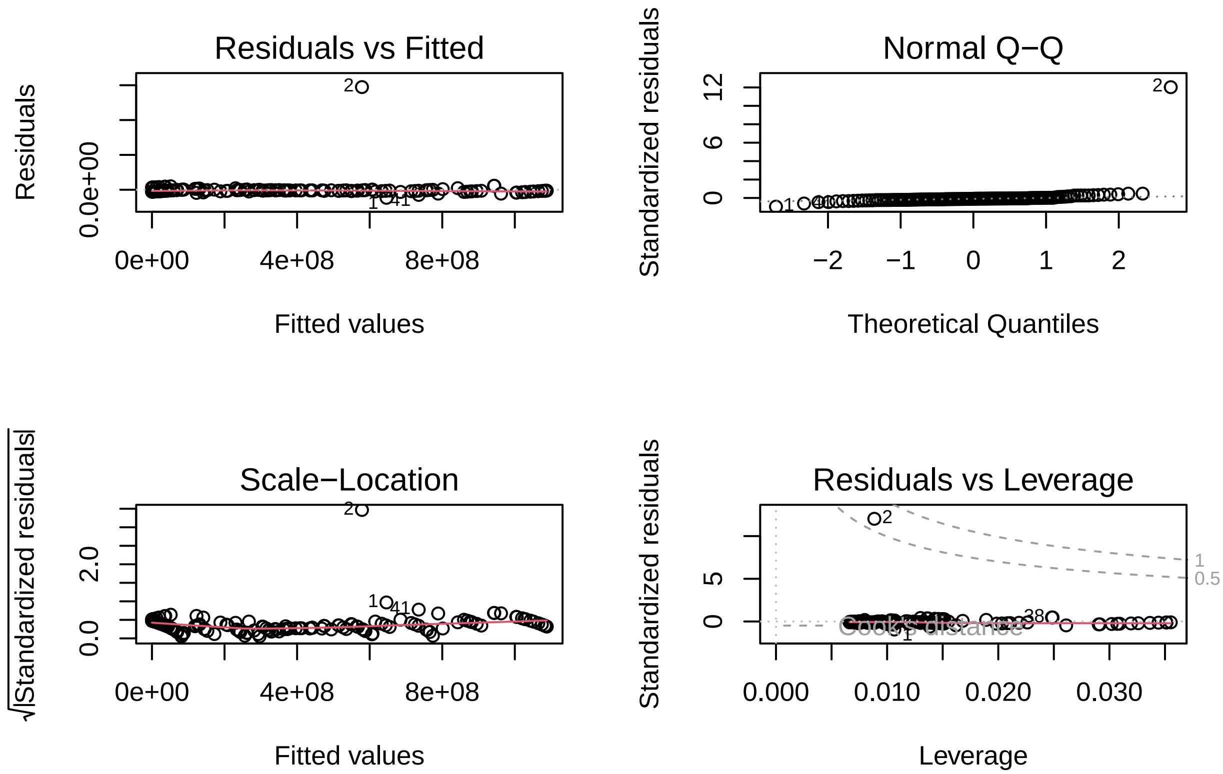
Could use some rounding.

```
## Warning in summary.lm(fit): essentially perfect fit: summary may be unreliable

##
## Call:
## lm(formula = ops ~ floor(-6 + 5.5 * size + 1.75 * (size^2)),
##     data = dat)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.152e-07 -1.989e-08 -1.040e-08 -4.010e-09  1.473e-06
##
## Coefficients:
##                                           Estimate Std. Error    t value
## (Intercept)                             -1.552e-07  1.514e-08 -1.025e+01
## floor(-6 + 5.5 * size + 1.75 * (size^2))  1.000e+00  2.967e-17  3.371e+16
##                                         Pr(>|t|)
## (Intercept)                               <2e-16 ***
## floor(-6 + 5.5 * size + 1.75 * (size^2))   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.23e-07 on 149 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 1.136e+33 on 1 and 149 DF,  p-value: < 2.2e-16
```

```
## Warning in summary.lm(object, ...): essentially perfect fit: summary may be
## unreliable
```

```
##                                                     2.5 %        97.5 %
## (Intercept)                                 -1.851428e-07 -1.252929e-07
## floor(-6 + 5.5 * size + 1.75 * (size^2))  1.000000e+00  1.000000e+00
```

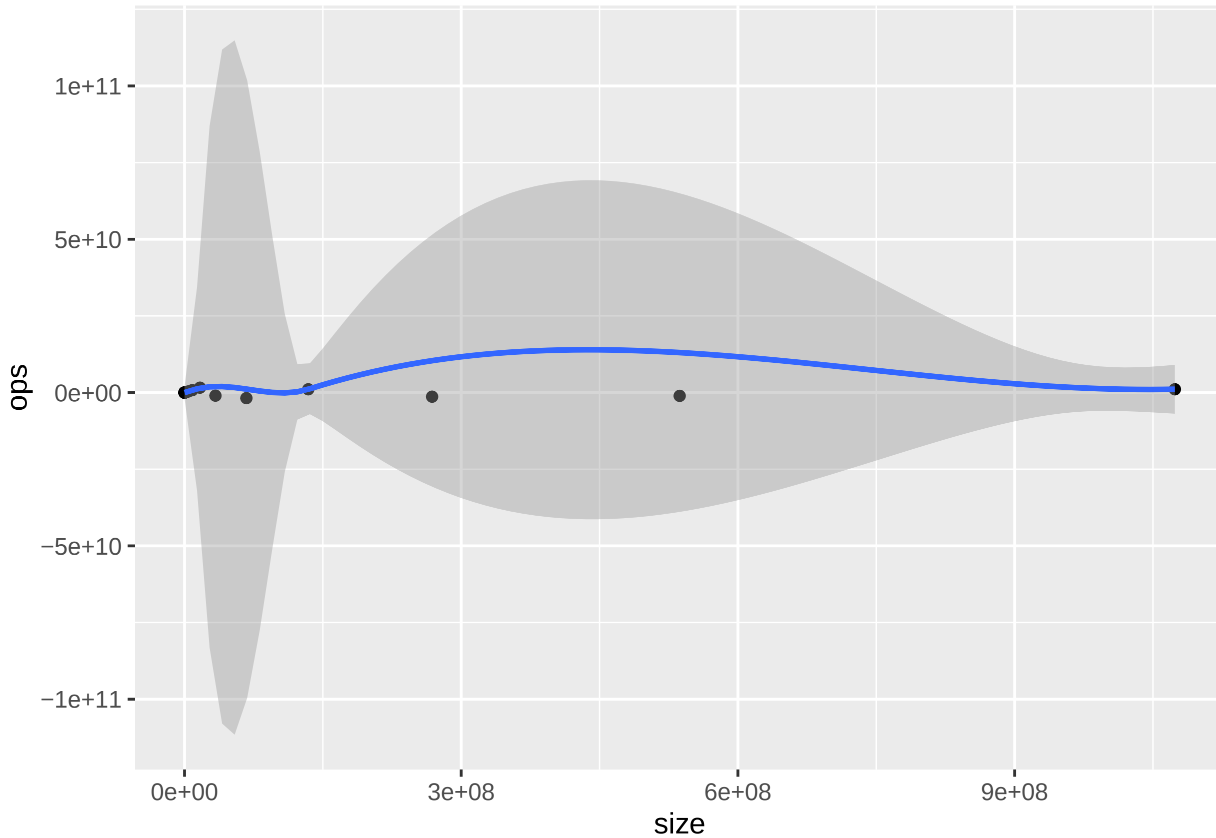


A perfect fit! With: $\hat{\beta}_0 = -6$, $\hat{\beta}_1 = 5.5$, $\hat{\beta}_2 = 1.75$

$$T(n) = \lfloor 1.75\ n^2 + 5.5\ n - 6 \rfloor$$

Question 5 - `pattern`

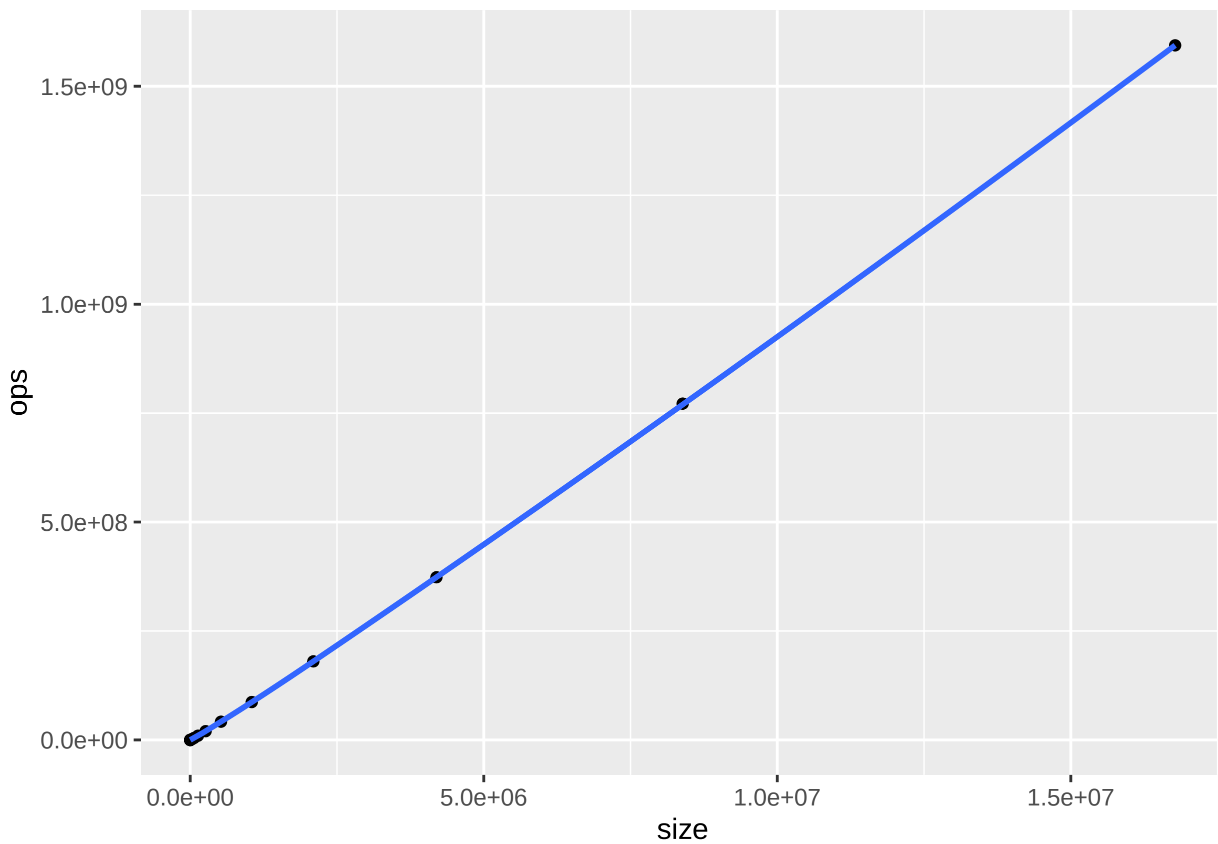
Make a scatterplot with a smooth line.




```
##       size                ops
##  Min.   :1.000e+00   Min.   :-1.812e+09
##  1st Qu.:1.920e+02   1st Qu.: 3.990e+02
##  Median :3.277e+04   Median : 1.147e+05
##  Mean   :6.927e+07   Mean   :-1.000e+01
##  3rd Qu.:6.291e+06   3rd Qu.: 3.106e+07
##  Max.   :1.074e+09   Max.   : 1.594e+09
```

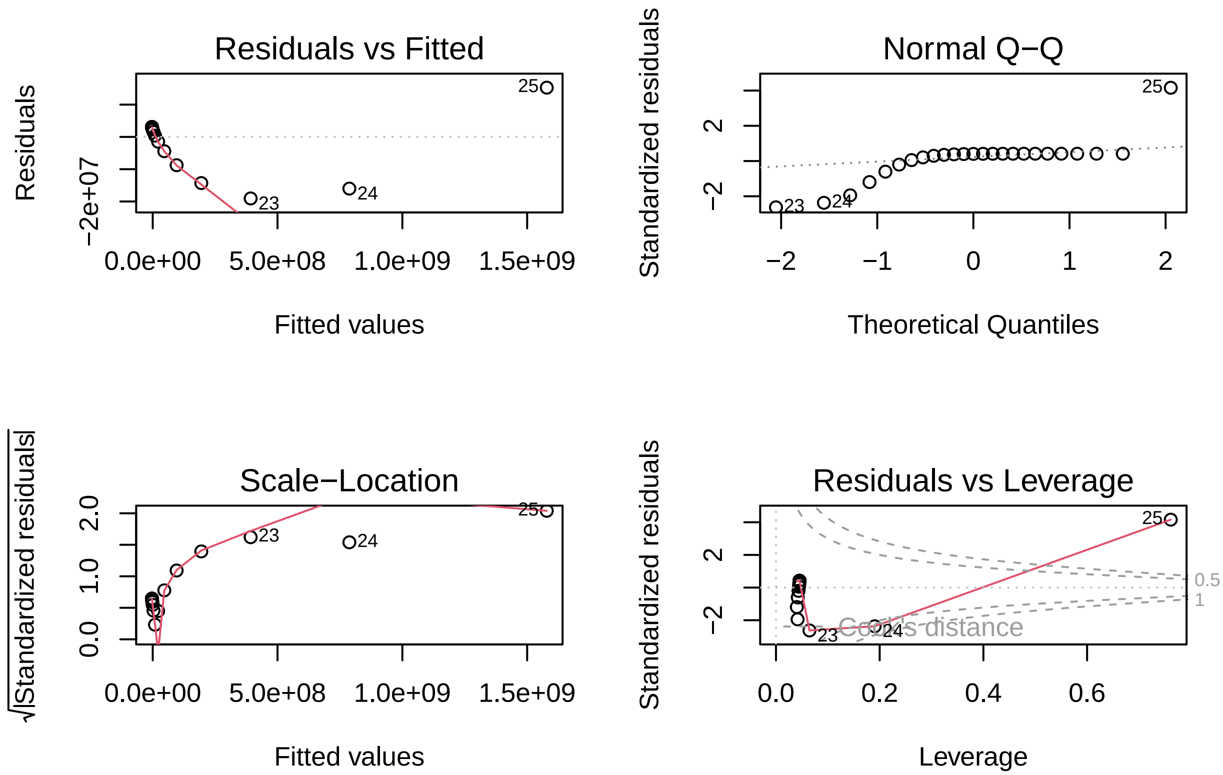
Oops! Operation count can't be negative (possibly because of integer overflow), thus need to remove those outliers. (This test program was compiled with `-fwrapv` in `g++`.)

```
##          size         ops
## 26   33554432 -1006632969
## 27   67108864 -1811939337
## 28  134217728  1073741815
## 29  268435456 -1342177289
## 30  536870912 -1073741833
## 31 1073741824  1073741815
```



Looks suspiciously linear. Let's try that.

```
##
## Call:
## lm(formula = ops ~ size, data = dat)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -19065020    393213   2972017   3048599  15252016
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.050e+06  1.601e+06  -1.906   0.0693 .
## size         9.427e+01  4.131e-01 228.190   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7508000 on 23 degrees of freedom
## Multiple R-squared:  0.9996, Adjusted R-squared:  0.9995
## F-statistic: 5.207e+04 on 1 and 23 DF,  p-value: < 2.2e-16
```



Residuals don't meet LR assumptions, suggesting a different model, possibly linearithmic. Trying that:

$$\mu_y = \beta_0 + \beta_1 x + \beta_2 \log_2 x + \beta_3 x \log_2 x + \epsilon$$

```
## Warning in summary.lm(fit): essentially perfect fit: summary may be unreliable

##
## Call:
## lm(formula = ops ~ size * I(log2(size)), data = dat)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -3.892e-08 -2.377e-08 -4.159e-09  9.228e-09  1.850e-07
##
## Coefficients:
##                      Estimate Std. Error    t value Pr(>|t|)
## (Intercept)        -9.000e+00  1.971e-08 -4.566e+08   <2e-16 ***
## size                2.300e+01  1.115e-13  2.063e+14   <2e-16 ***
## I(log2(size))      -3.751e-09  1.869e-09 -2.007e+00   0.0578 .
## size:I(log2(size))  3.000e+00  4.639e-15  6.467e+14   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.478e-08 on 21 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 4.882e+32 on 3 and 21 DF,  p-value: < 2.2e-16
```

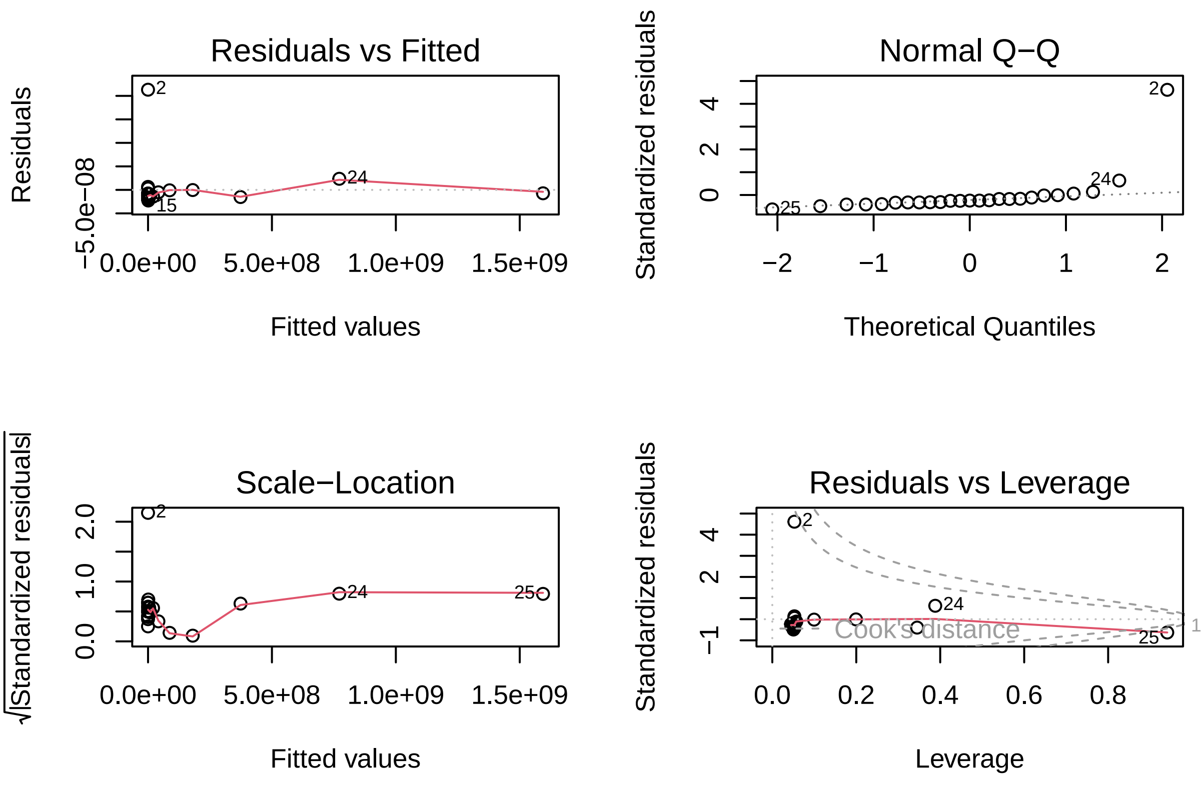
$\log_2 x$ alone is insignificant, dropping it.

$$\mu_y = \beta_0 + \beta_1 x + \beta_2 x \log_2 x + \epsilon$$

```
##
## Call:
## lm(formula = ops ~ size + I(size * log2(size)), data = dat)
##
## Coefficients:
##          (Intercept)                   size  I(size * log2(size))
##                   -9                     23                     3
```

```
## Warning in summary.lm(object, ...): essentially perfect fit: summary may be
## unreliable
```

```
##                        2.5 % 97.5 %
## (Intercept)               -9     -9
## size                      23     23
## I(size * log2(size))       3      3
```

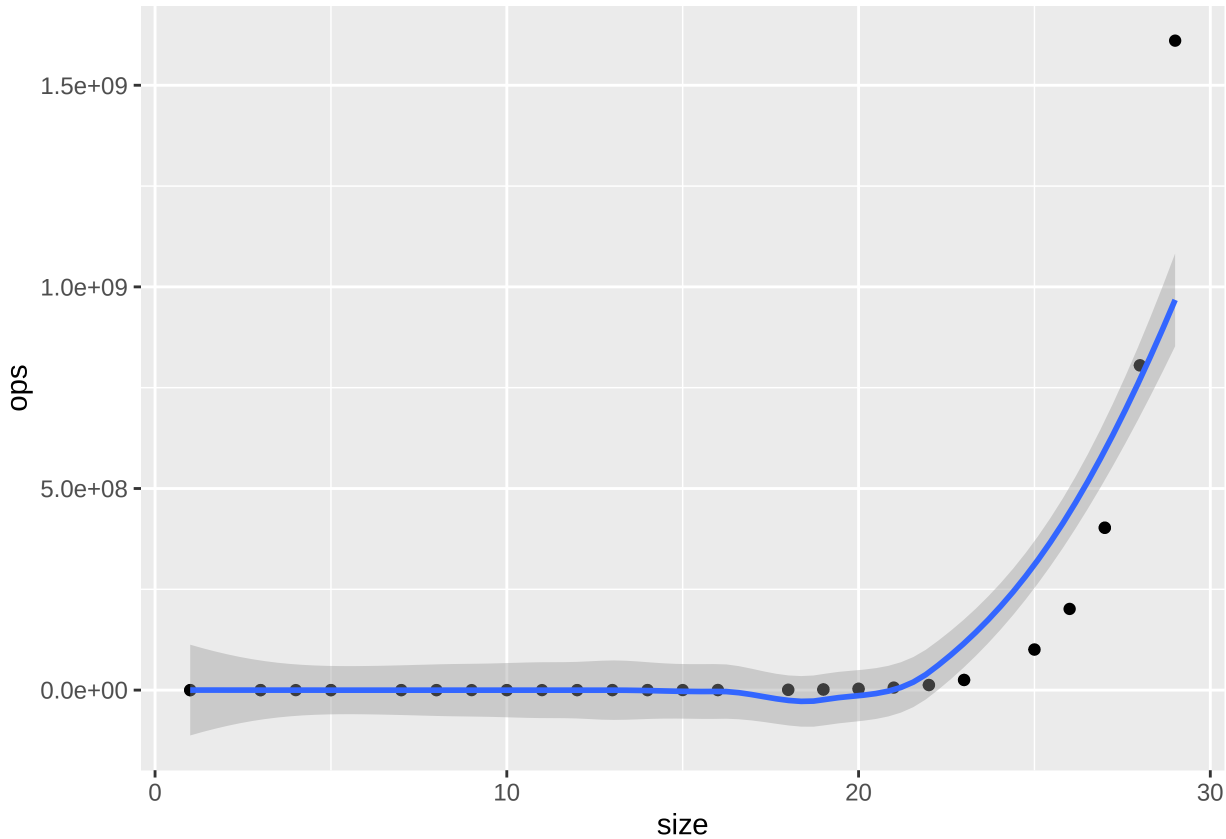


Perfect fit with: $\hat{\beta}_0 = -9$, $\hat{\beta}_1 = 23$, $\hat{\beta}_2 = 3$

$$T(n) = 3n \log_2 n + 23n - 9$$

Question 6 - `lsearch`

Make a scatterplot with a smooth line.




```
##       size            ops
##  Min.   : 1.00   Min.   :2.000e+00
##  1st Qu.: 8.00   1st Qu.:7.640e+02
##  Median :16.00   Median :1.966e+05
##  Mean   :15.27   Mean   :1.125e+08
##  3rd Qu.:22.00   3rd Qu.:1.258e+07
##  Max.   :29.00   Max.   :1.611e+09
```

Clearly exponential, checking for interaction with polynomials.

$$\mu_y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 2^x + \beta_4 \; x \cdot 2^x + \beta_5 \; x^2 \cdot 2^x + \epsilon$$

```
## Warning in summary.lm(fit): essentially perfect fit: summary may be unreliable
```

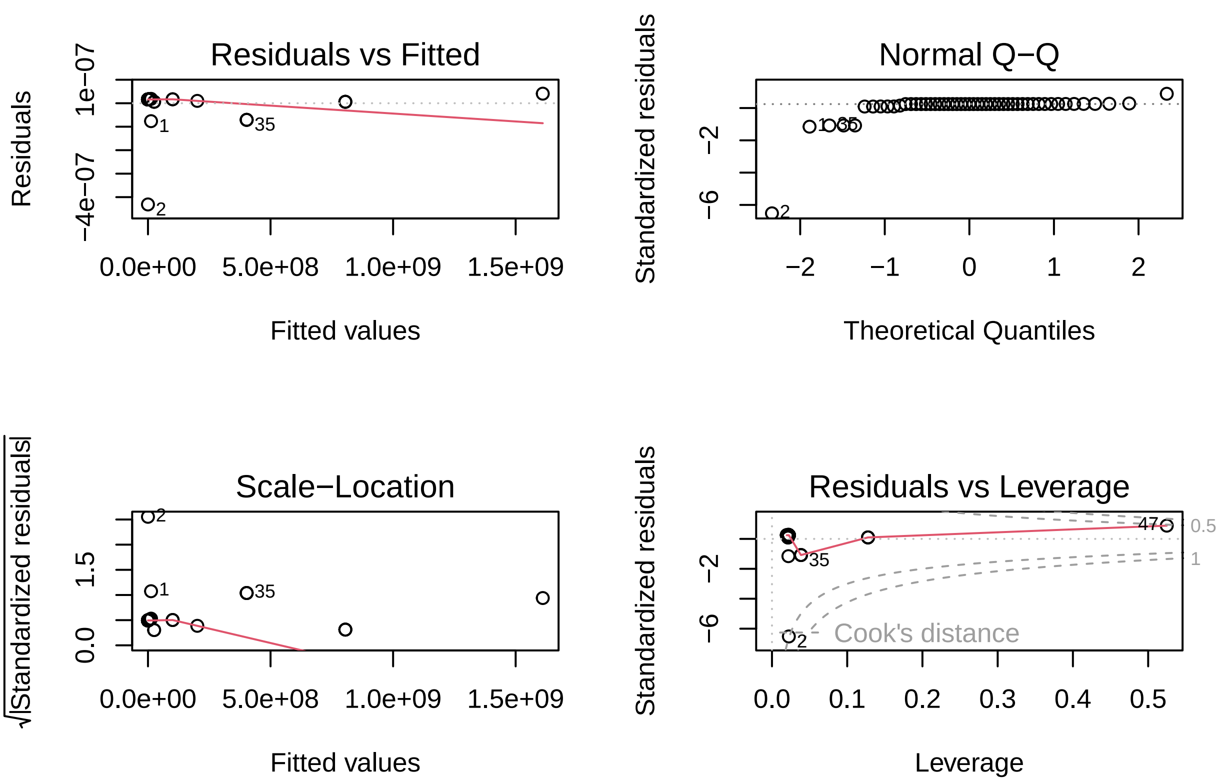
```
##
## Call:
## lm(formula = ops ~ poly(size, 2) * I(2^size), data = dat)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -5.468e-07 -7.360e-09  1.900e-08  3.553e-08  7.101e-08
```

```
##
## Coefficients:
##                          Estimate Std. Error    t value Pr(>|t|)
## (Intercept)            -4.000e+00  4.037e-08 -9.907e+07  < 2e-16 ***
## poly(size, 2)1          9.537e-07  3.536e-07  2.697e+00 0.009804 **
## poly(size, 2)2          9.537e-07  2.647e-07  3.603e+00 0.000782 ***
## I(2^size)               3.000e+00  6.584e-14  4.557e+13  < 2e-16 ***
## poly(size, 2)1:I(2^size) -3.925e-14  4.353e-13 -9.000e-02 0.928561
## poly(size, 2)2:I(2^size)  8.302e-15  1.209e-13  6.900e-02 0.945544
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.076e-07 on 45 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 7.676e+31 on 5 and 45 DF,  p-value: < 2.2e-16
```

Last 2 are insignificant, coefficients of polynomials are near zero, dropping them all.

$$\mu_y = \beta_0 + \beta_1 2^x + \epsilon$$

```
##
## Call:
## lm(formula = ops ~ I(2^size), data = dat)
##
## Coefficients:
## (Intercept)    I(2^size)
##          -4            3
```



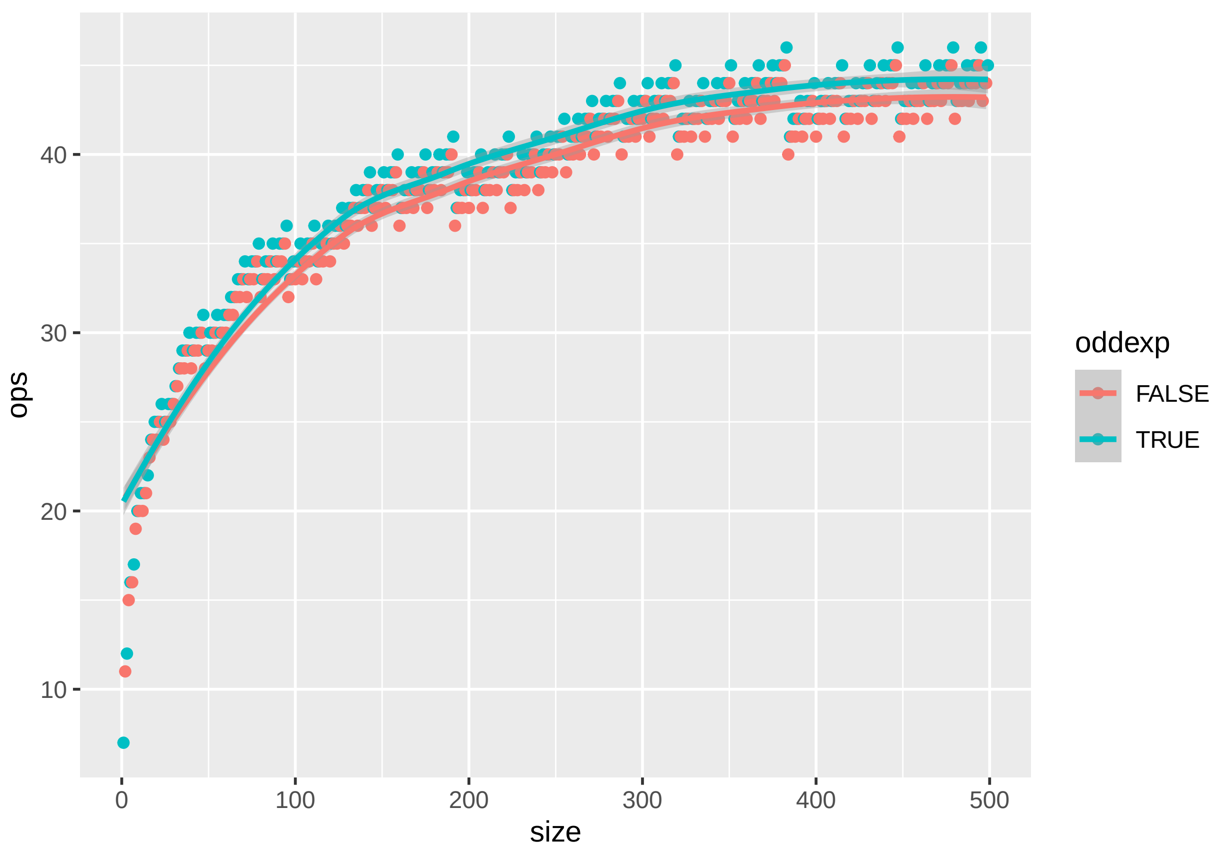
Perfect fit with: $\hat{\beta}_0 = -4$, $\hat{\beta}_1 = 3$

$$T(n) = 3 \cdot 2^n - 4$$

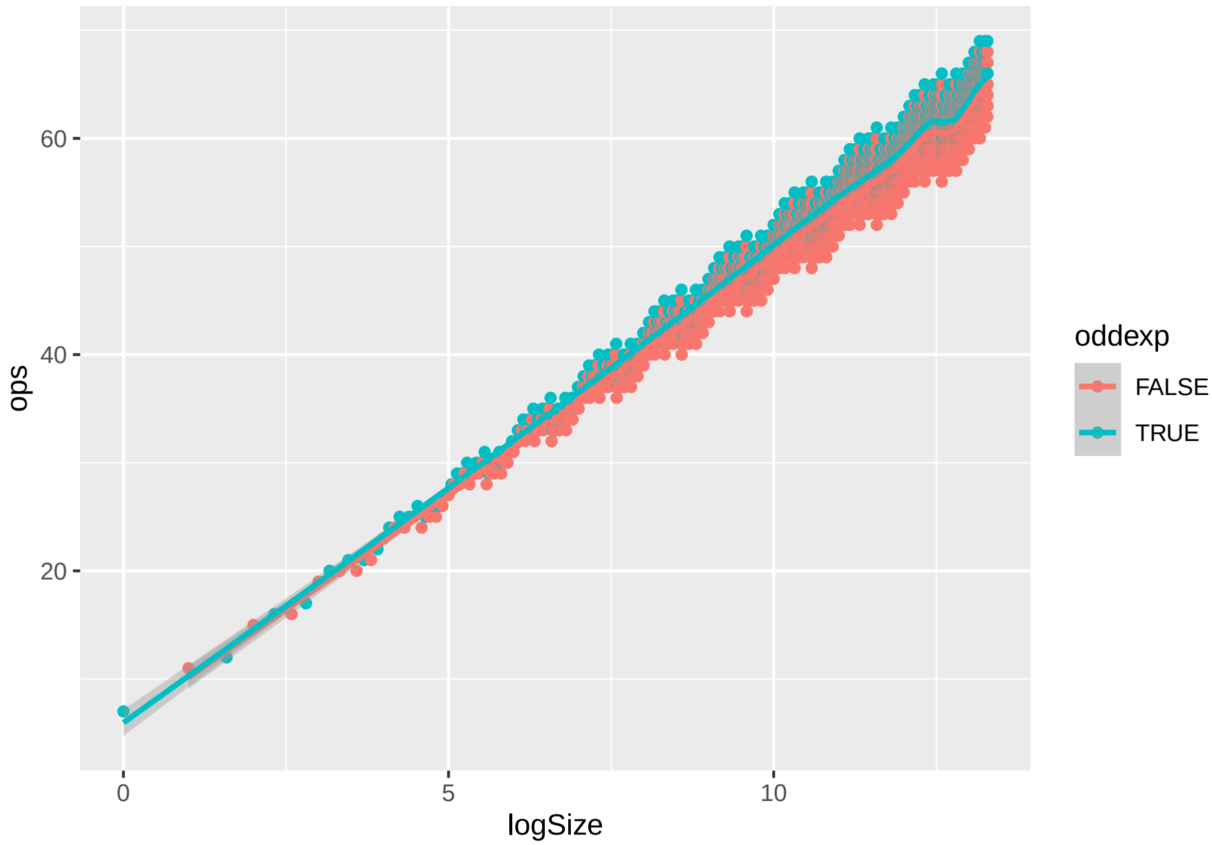
Question 7 - `pow`

Make a scatterplot with a smooth line. Odd exponents cost more operations than even ones, so make them stand out.

```
##       size            ops           oddexp           logSize
##  Min.   :    1   Min.   : 7.00   Mode :logical   Min.   : 0.00
##  1st Qu.: 2501   1st Qu.:55.00   FALSE:5000      1st Qu.:11.29
##  Median : 5000   Median :60.00   TRUE :5000      Median :12.29
##  Mean   : 5000   Mean   :57.91                   Mean   :11.85
##  3rd Qu.: 7500   3rd Qu.:62.00                   3rd Qu.:12.87
##  Max.   :10000   Max.   :69.00                   Max.   :13.29
##      logOps
##  Min.   :2.807
##  1st Qu.:5.781
##  Median :5.907
##  Mean   :5.844
##  3rd Qu.:5.954
##  Max.   :6.109
```



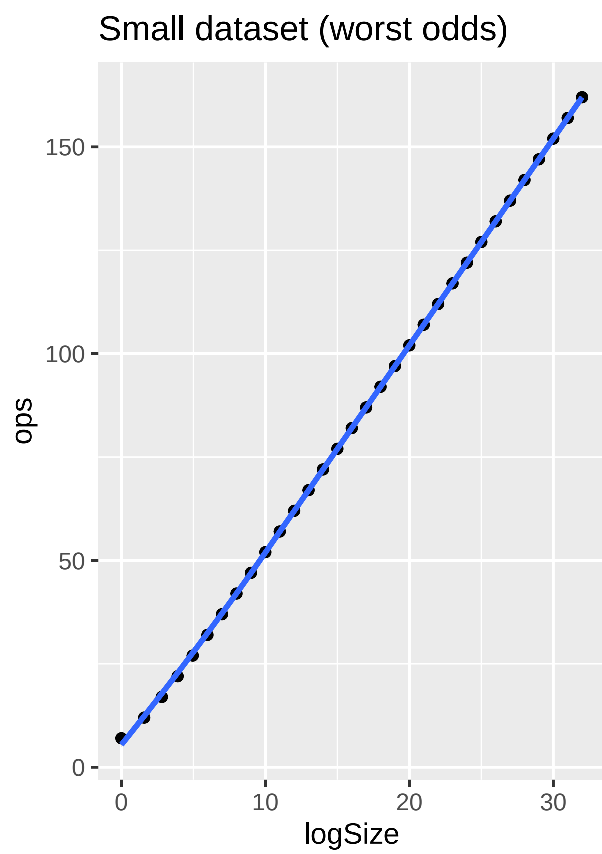
The general trend is logarithmic, but has a periodic pattern to it. Trying a logarithmic transformation on the predictor variable.

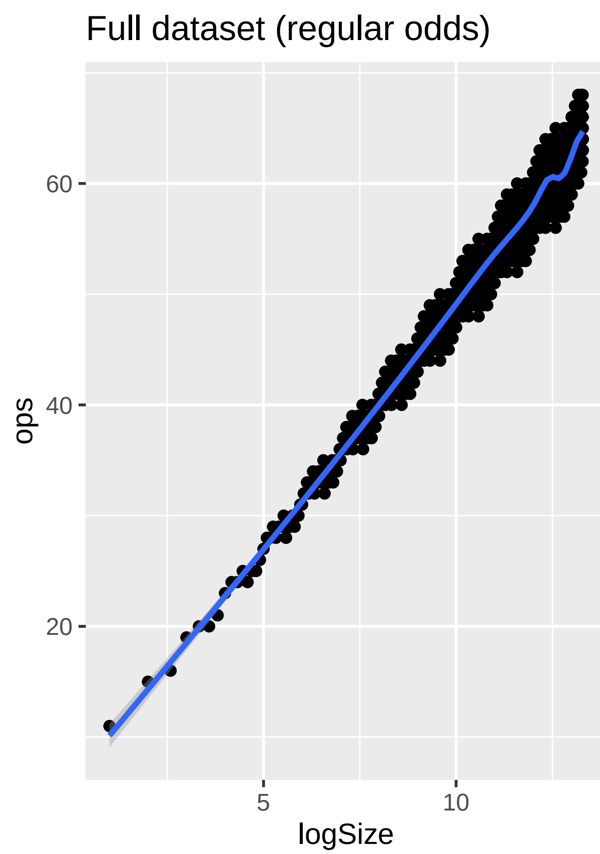


Odd exponents clearly cost more, but the data is heteroscedastic! As we are more interested in the upper bound, trying with a small dataset of only odd exponents where their binary representations are all 1s.

```
##   size ops  logSize   logOps
## 1    1   7 0.000000 2.807355
## 2    3  12 1.584963 3.584963
## 3    7  17 2.807355 4.087463
## 4   15  22 3.906891 4.459432
## 5   31  27 4.954196 4.754888
## 6   63  32 5.977280 5.000000
```

```
##       size                ops             logSize           logOps
##  Min.   :1.000e+00   Min.   :  7.00   Min.   : 0.000   Min.   :2.807
##  1st Qu.:4.470e+02   1st Qu.: 45.75   1st Qu.: 8.746   1st Qu.:5.514
##  Median :9.830e+04   Median : 84.50   Median :16.500   Median :6.400
##  Mean   :2.684e+08   Mean   : 84.50   Mean   :16.444   Mean   :6.067
##  3rd Qu.:2.097e+07   3rd Qu.:123.25   3rd Qu.:24.250   3rd Qu.:6.945
##  Max.   :4.295e+09   Max.   :162.00   Max.   :32.000   Max.   :7.340
```





Trying to fit a logarithm to that small dataset.

$$\mu_y = \beta_0 + \beta_1 \log_2 x + \epsilon$$

```
##
## Call:
## lm(formula = ops ~ logSize, data = sodat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.6884 -0.4606 -0.1297  0.2023  3.9113
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.08872    0.30189   10.23 2.68e-11 ***
## logSize      4.95082    0.01597  310.00  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8423 on 30 degrees of freedom
## Multiple R-squared:  0.9997, Adjusted R-squared:  0.9997
## F-statistic: 9.61e+04 on 1 and 30 DF,  p-value: < 2.2e-16
```

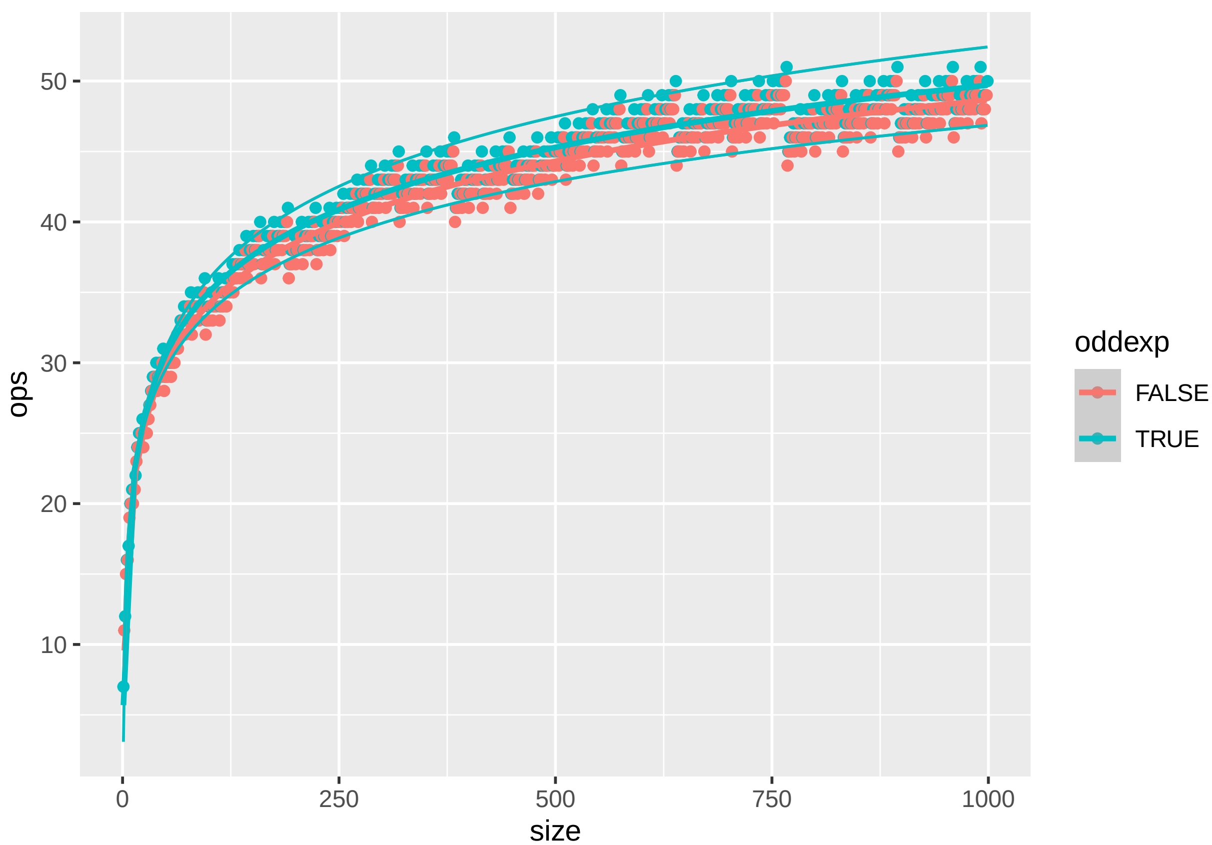
```
##                2.5 %   97.5 %
## (Intercept) 2.472178 3.705271
## logSize     4.918203 4.983435
```

Model suggests the average operation count of the worst odd exponents can use $5 \log_2 n + 3$. While we're at it, also try with even exponents that have binary representations starting with 1 and rest with 0s. Finding these two can help inform where the bounds are.

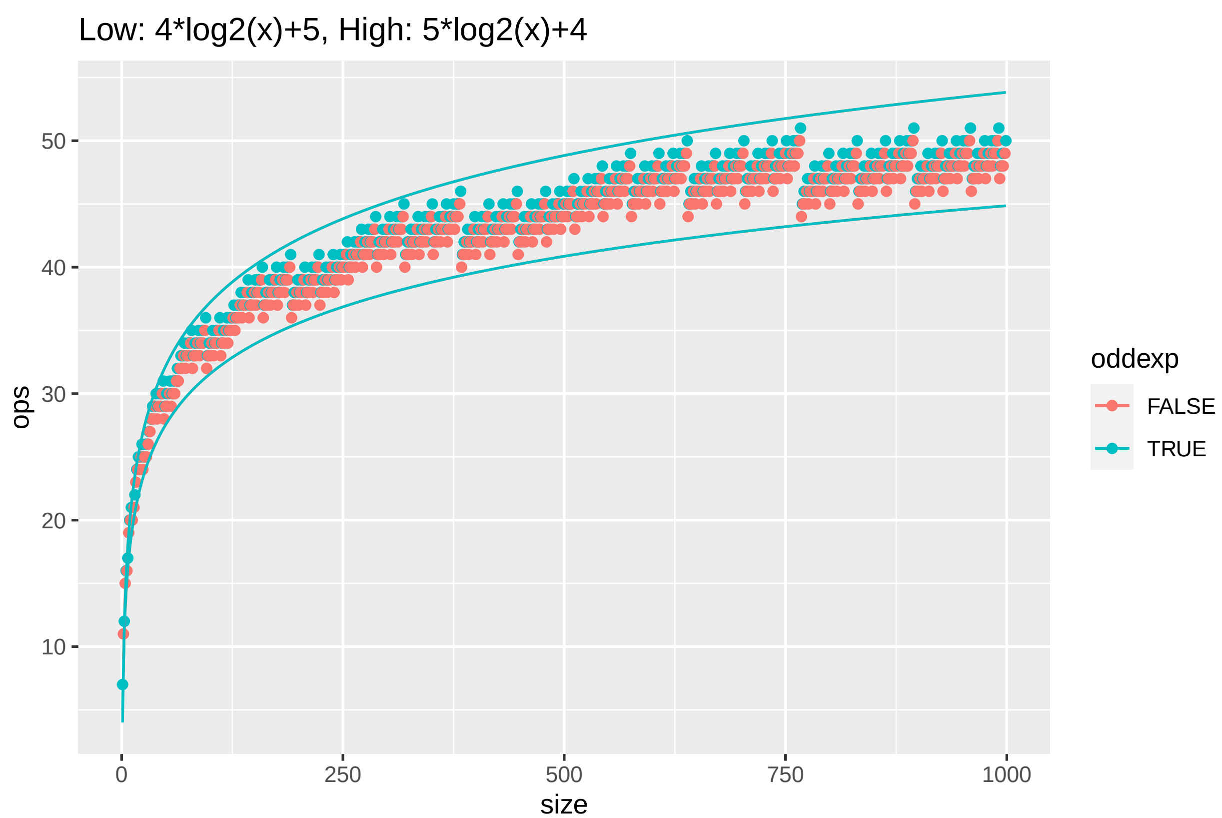
```
##   size ops logSize  logOps
## 1    2  11       1 3.459432
## 2    4  15       2 3.906891
## 3    8  19       3 4.247928
## 4   16  23       4 4.523562
## 5   32  27       5 4.754888
## 6   64  31       6 4.954196
```

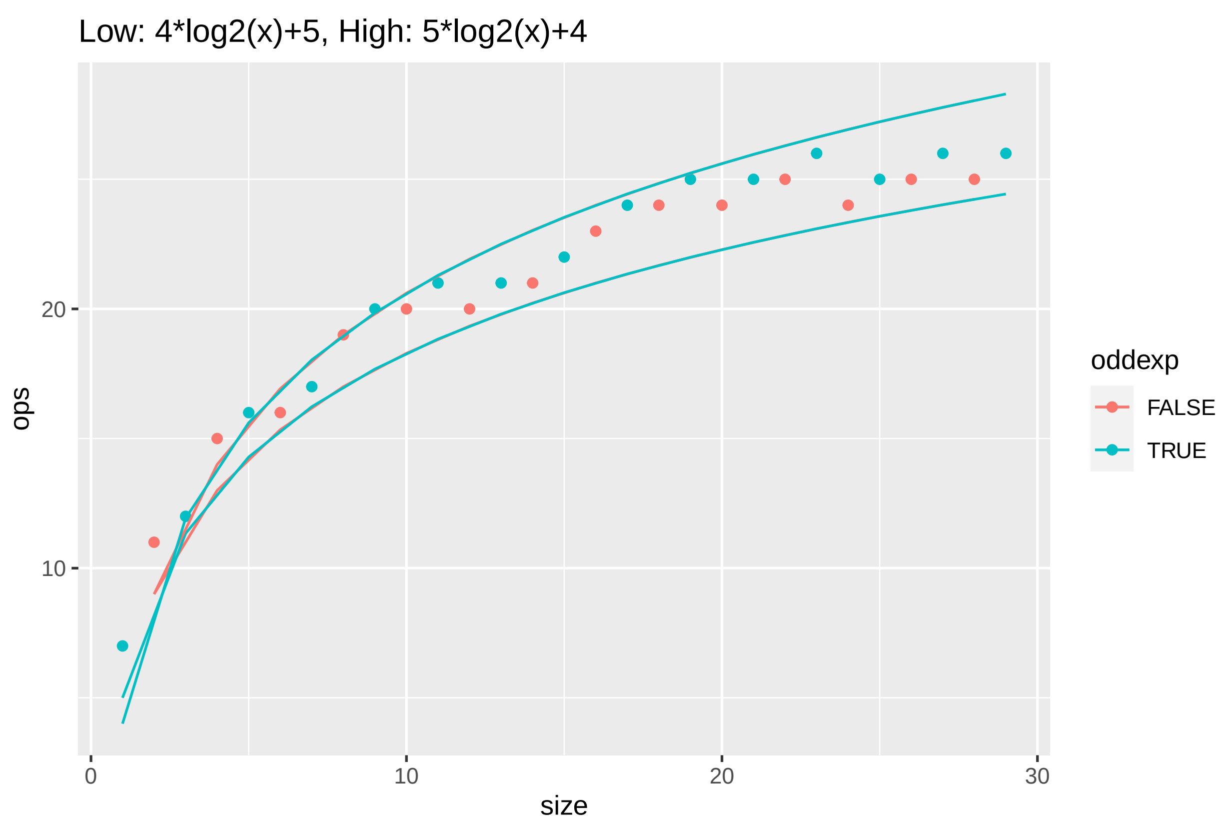
```
##
## Call:
## lm(formula = ops ~ logSize, data = sedat)
##
## Coefficients:
## (Intercept)      logSize
##           7            4
```

Model suggests $4 \log_2 n + 7$ for the best even exponents. Plotting these two against the original dataset.

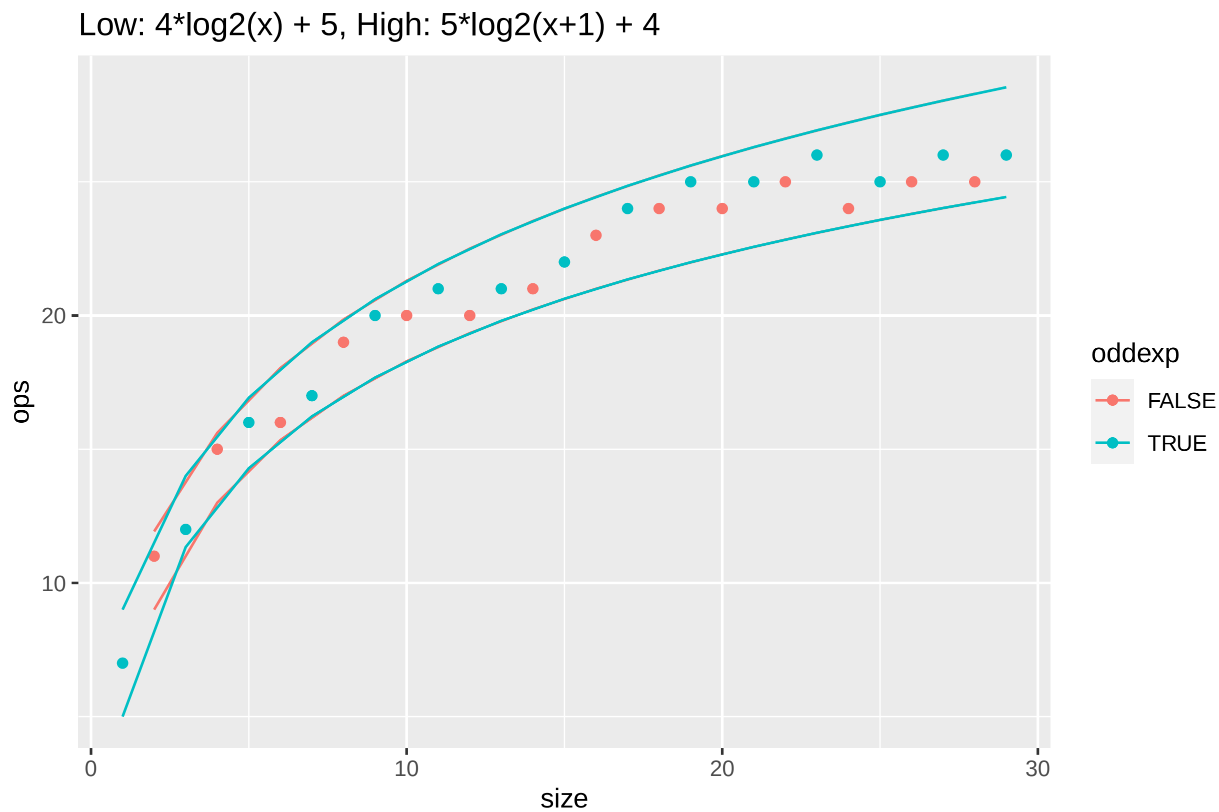


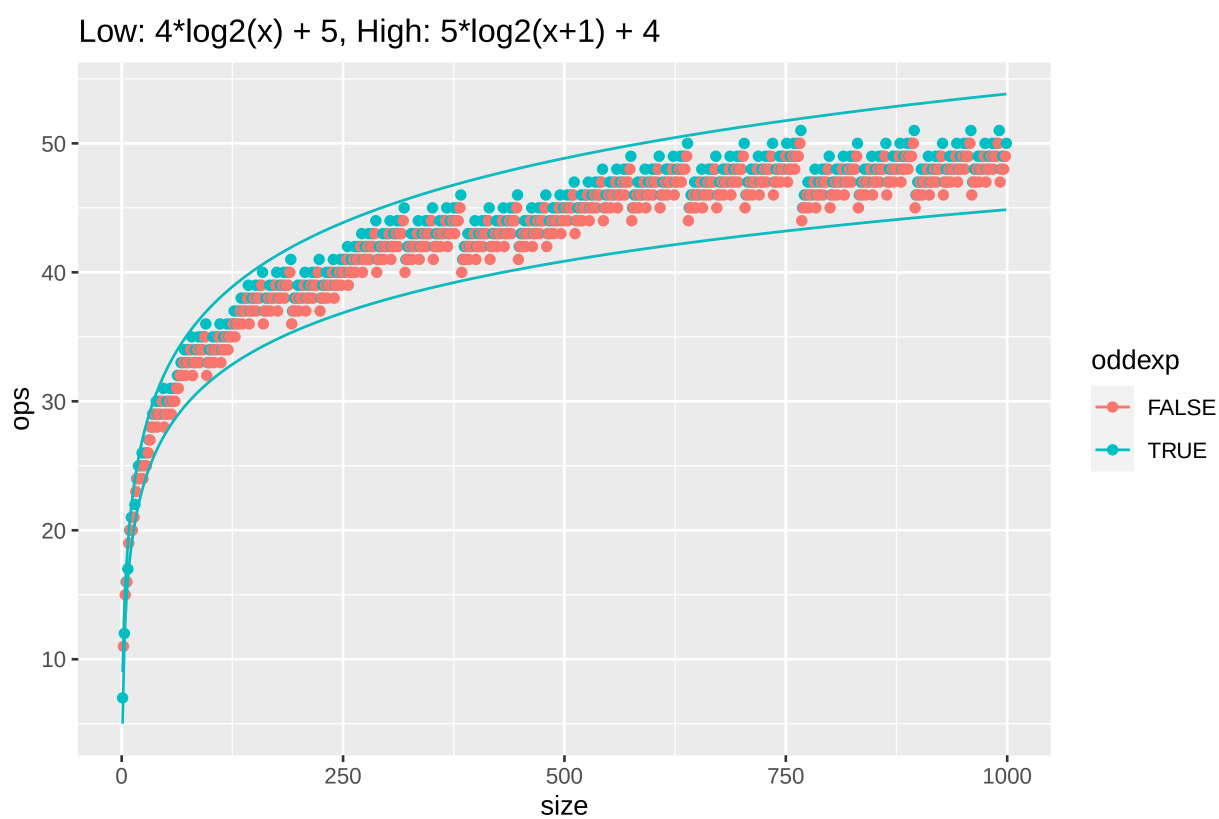
As they apparently aren't the best nor worst cases, fine-tuning is needed. Trying to keep the coefficients on $\log_2 x$ ($\hat{\beta}_1$) the same, shifting vertically first, and then horizontally as needed.



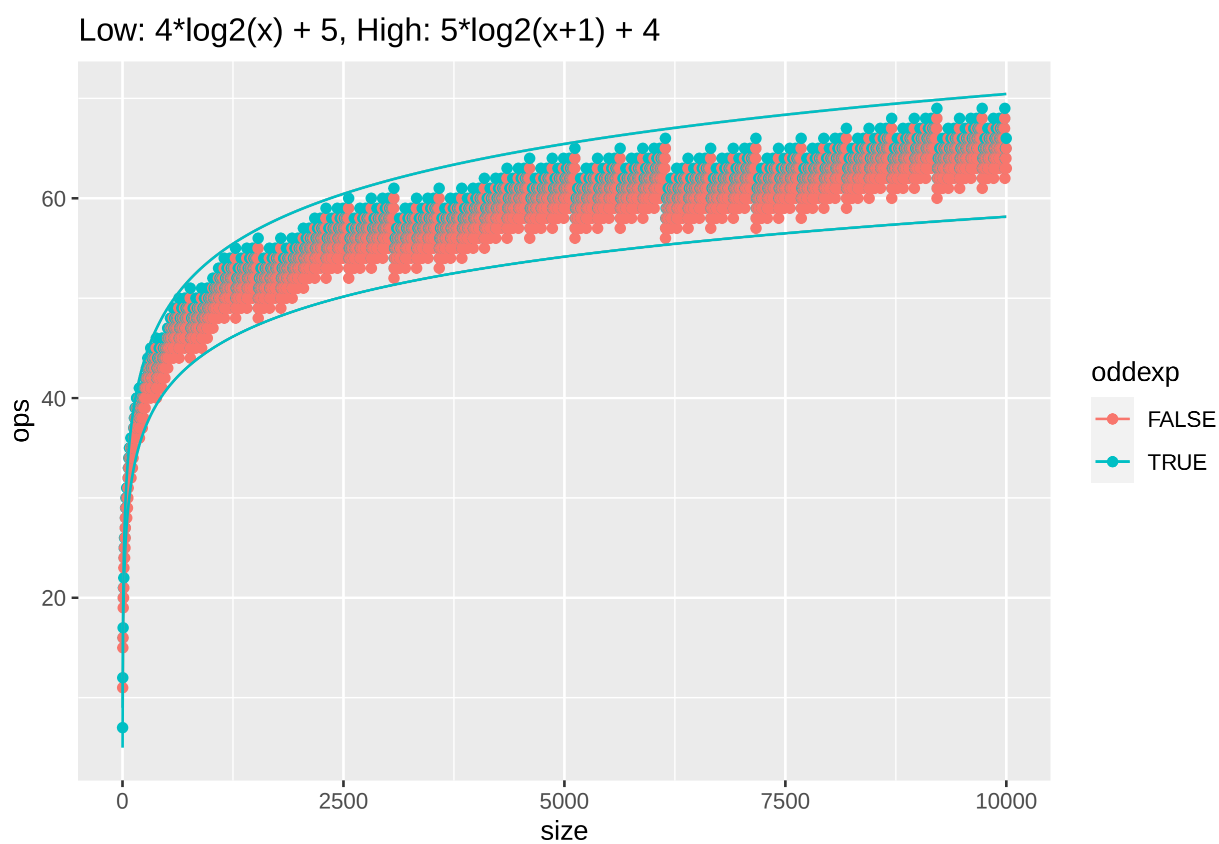


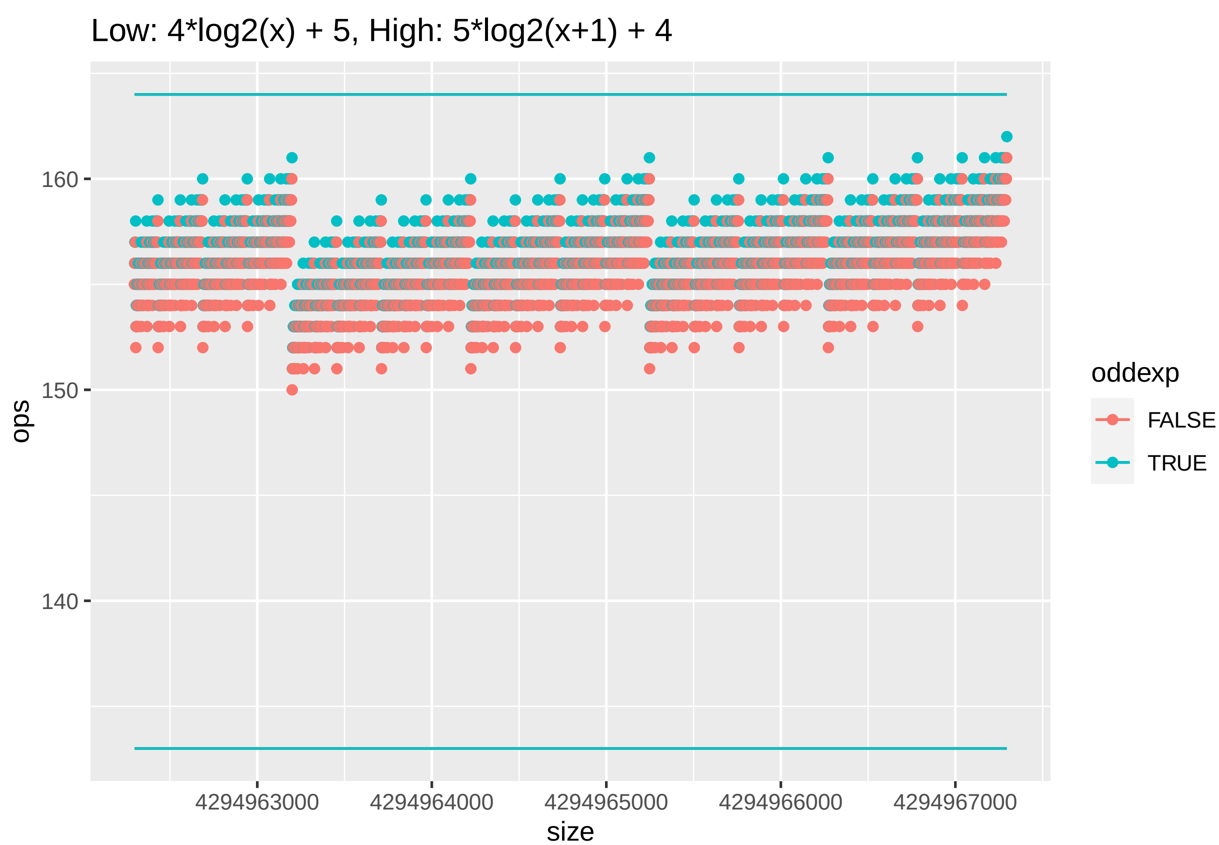
Not good. Shifting the upper bound to the left by 1.





Looking great! Now trying with larger datasets.


Low: 4*log2(x) + 5, High: 5*log2(x+1) + 4
ops
60
40
20
0
2500
5000
7500
10000
size
oddexp
FALSE
TRUE


Low: 4*log2(x) + 5, High: 5*log2(x+1) + 4
ops
160
150
140
4294963000
4294964000
4294965000
4294966000
4294967000
size
oddexp
FALSE
TRUE

Computing for datapoints outside these bounds in the above two:

```
## [1] size    ops     oddexp  logSize logOps  lowfit  highfit
## <0 rows> (or 0-length row.names)
```

```
## [1] size    ops     oddexp  logSize logOps  lowfit  highfit
## <0 rows> (or 0-length row.names)
```

Good enough for me!

$$4\log_2(n) + 5 \leq T(n) \leq 5\log_2(x+1) + 4$$

Appendix B - Samples Used

Previews of the larger datasets.

Question 4 - `ssort`

```
##    input.size..n. operation.count
## 1           19215       646234070
## 2           18184       578751254
## 3           21031       774145846
## 4            5038        44445230
## 5            5841        59737361
## 6           17494       535666274
## 7           24777      1074460793
## 8           20202       714322512
## 9            7044        86870124
## 10          22276       868507820
## 11           6814        81291014
## 12           2989        15651145
## 13           4615        37297270
## 14          14602       373212512
## 15          13967       341461718
## 16          20780       755778984
## 17          20349       724755065
## 18          17355       527188490
## 19          19020       633185304
## 20          24876      1083063720
## 21           9342       152779062
## 22          19189       644486545
## 23           1340         3149664
## 24          17154       515048844
## 25           2673        12518321
## 26          24510      1051429974
## 27          15245       406801385
## 28           4937        42681593
## 29          16483       475546906
## 30          18612       606313812
```

Question 5 - `pattern`

```
##    input.size..n. operation.count
## 1               1              14
## 2               2              43
## 3               4             107
## 4               8             247
## 5              16             551
## 6              32            1207
## 7              64            2615
## 8             128            5623
## 9             256           12023
## 10            512           25591
## 11           1024           54263
## 12           2048          114679
## 13           4096          241655
## 14           8192          507895
## 15          16384         1064951
## 16          32768         2228215
## 17          65536         4653047
## 18         131072         9699319
## 19         262144        20185079
## 20         524288        41943031
## 21        1048576        87031799
## 22        2097152       180355063
## 23        4194304       373293047
## 24        8388608       771751927
## 25       16777216      1593835511
## 26       33554432     -1006632969
## 27       67108864     -1811939337
## 28      134217728      1073741815
## 29      268435456     -1342177289
## 30      536870912     -1073741833
```

Question 6 - `lsearch`

```
##    input.size..n. operation.count
## 1              22        12582908
## 2              10            3068
## 3              19         1572860
## 4              28       805306364
## 5               7             380
## 6               9            1532
## 7               4              44
## 8              13           24572
## 9              21         6291452
## 10              5              92
## 11             20         3145724
## 12              3              20
## 13              1               2
## 18             12           12284
## 21             26       201326588
## 23             18          786428
```

```
## 24            25       100663292
## 25            23        25165820
## 26            15           98300
## 29            16          196604
## 32             8             764
## 35            27       402653180
## 42            14           49148
## 47            29      1610612732
## 49            11            6140
```

Question 7 - `pow`

In order: 1. Original dataset. 2. Worst odd exponents. 3. Best even exponents. 4. Extreme dataset.

```
##    size ops oddexp  logSize   logOps
## 1     1   7   TRUE 0.000000 2.807355
## 2     2  11  FALSE 1.000000 3.459432
## 3     3  12   TRUE 1.584963 3.584963
## 4     4  15  FALSE 2.000000 3.906891
## 5     5  16   TRUE 2.321928 4.000000
## 6     6  16  FALSE 2.584963 4.000000
## 7     7  17   TRUE 2.807355 4.087463
## 8     8  19  FALSE 3.000000 4.247928
## 9     9  20   TRUE 3.169925 4.321928
## 10   10  20  FALSE 3.321928 4.321928
## 11   11  21   TRUE 3.459432 4.392317
## 12   12  20  FALSE 3.584963 4.321928
## 13   13  21   TRUE 3.700440 4.392317
## 14   14  21  FALSE 3.807355 4.392317
## 15   15  22   TRUE 3.906891 4.459432
## 16   16  23  FALSE 4.000000 4.523562
## 17   17  24   TRUE 4.087463 4.584963
## 18   18  24  FALSE 4.169925 4.584963
## 19   19  25   TRUE 4.247928 4.643856
## 20   20  24  FALSE 4.321928 4.584963
## 21   21  25   TRUE 4.392317 4.643856
## 22   22  25  FALSE 4.459432 4.643856
## 23   23  26   TRUE 4.523562 4.700440
## 24   24  24  FALSE 4.584963 4.584963
## 25   25  25   TRUE 4.643856 4.643856
## 26   26  25  FALSE 4.700440 4.643856
## 27   27  26   TRUE 4.754888 4.700440
## 28   28  25  FALSE 4.807355 4.643856
## 29   29  26   TRUE 4.857981 4.700440
## 30   30  26  FALSE 4.906891 4.700440
```

```
##       size ops  logSize   logOps
## 1        1   7 0.000000 2.807355
## 2        3  12 1.584963 3.584963
## 3        7  17 2.807355 4.087463
## 4       15  22 3.906891 4.459432
## 5       31  27 4.954196 4.754888
## 6       63  32 5.977280 5.000000
```

```
## 7       127  37  6.988685 5.209453
## 8       255  42  7.994353 5.392317
## 9       511  47  8.997179 5.554589
## 10     1023  52  9.998590 5.700440
## 11     2047  57 10.999295 5.832890
## 12     4095  62 11.999648 5.954196
## 13     8191  67 12.999824 6.066089
## 14    16383  72 13.999912 6.169925
## 15    32767  77 14.999956 6.266787
## 16    65535  82 15.999978 6.357552
## 17   131071  87 16.999989 6.442943
## 18   262143  92 17.999994 6.523562
## 19   524287  97 18.999997 6.599913
## 20 1048575 102 19.999999 6.672425

##        size ops logSize   logOps
## 1         2  11       1 3.459432
## 2         4  15       2 3.906891
## 3         8  19       3 4.247928
## 4        16  23       4 4.523562
## 5        32  27       5 4.754888
## 6        64  31       6 4.954196
## 7       128  35       7 5.129283
## 8       256  39       8 5.285402
## 9       512  43       9 5.426265
## 10     1024  47      10 5.554589
## 11     2048  51      11 5.672425
## 12     4096  55      12 5.781360
## 13     8192  59      13 5.882643
## 14    16384  63      14 5.977280
## 15    32768  67      15 6.066089
## 16    65536  71      16 6.149747
## 17   131072  75      17 6.228819
## 18   262144  79      18 6.303781
## 19   524288  83      19 6.375039
## 20  1048576  87      20 6.442943

##            size ops oddexp logSize   logOps lowfit highfit
## 4986 4294967281 159   TRUE      32 7.312883    133     164
## 4987 4294967282 159  FALSE      32 7.312883    133     164
## 4988 4294967283 160   TRUE      32 7.321928    133     164
## 4989 4294967284 159  FALSE      32 7.312883    133     164
## 4990 4294967285 160   TRUE      32 7.321928    133     164
## 4991 4294967286 160  FALSE      32 7.321928    133     164
## 4992 4294967287 161   TRUE      32 7.330917    133     164
## 4993 4294967288 159  FALSE      32 7.312883    133     164
## 4994 4294967289 160   TRUE      32 7.321928    133     164
## 4995 4294967290 160  FALSE      32 7.321928    133     164
## 4996 4294967291 161   TRUE      32 7.330917    133     164
## 4997 4294967292 160  FALSE      32 7.321928    133     164
## 4998 4294967293 161   TRUE      32 7.330917    133     164
## 4999 4294967294 161  FALSE      32 7.330917    133     164
## 5000 4294967295 162   TRUE      32 7.339850    133     164
```