1. 1. A method to solve this would take as input an array of size ***n*** representing the values of ***a*** and a value of ***x***. It would loop from ***i=***0 to ***n***, accessing ***a*** at each element and multiplying that to the ***x***^***i***. The result at each step would then be added to a variable ***p***, which is returned at the end.

Accessing ***a*** each time can be done in constant time, while computing ***x***^***i*** is done, at the very end, in ***n*** time, therefore that step is considered O(n). Additionally, adding the result is done in constant time too. Because it is in a loop, that step is run ***n*** times, making the array accesses O(n), the exponent computation O(n^2), and updating the result would be O(n). The time complexity of this method is therefore O(n) + O(n^2) + O(n), which can be simplified to just O(n^2).

* 1. In the nested form, p(x) has a number of additions and multiplications that scale linearly with ***n***. Therefore, this method of evaluation is O(n).

Total time:

T(n) = {1 if n =1, T(n-1) + 2 otherwise}

O(n)



Pivot Selection

Divide S into groups of 3

[n/3] groups of size 1

O(1) time

Sort each group of size 3

Uses at most 3 comparisons

[n/3] \* 3 ~= n time

Determine median of each group

Pick the middle element of each group: O(1)

Gather all medians in a sequence: ~n time

Recurse

T(n/3) time

Recursive call

Guarantee [n/3] smaller, similar larger

Worst case, n/3 elements in L and 2n/3 in G

Takes T(2n/3) time

Pivot Selection 2n + T(n/3)

Partition n

Recursive Call T(2n/3)

T(n) = 3n + T(n/3) + T(2n/3)

Check T(n) <= cn

T(n) = 3n + T(n/3) + T(2n/3)

<= 3n + cn/3 + 2cn/3

9n + cn + 2cn <= 3cn

9 + 3c <= 3c

Since c is constrained to be greater than 0, a c does not exist such that T(n) <= cn. Therefore, by using subsequences of 3, the linear select algorithm would no longer be O(n).



**Algorithm** inPlaceLinearSelect(𝑆,𝑎,𝑏,𝑘)

**Input**: An array, 𝑆, of distinct elements; integers 𝑎 and 𝑏 such 𝑎 ≤ 𝑏; and integer 𝑘∈[𝑎+1, 𝑏+1]

**Output**: The 𝑘th smallest element of 𝑆

if n = 1 then

return S

p <- pickCleverPivot(S)

partition(a,b,S,p)

if k < p then

return inPlaceLinearSelect(S,a,p,K)

else if k > p

return inPlaceLinearSelect(S,p+1,b,K)

else

return p

**Algorithm** pickCleverPivot(𝑆, a, b)

**Input**: An array, 𝑆, containing n elements; indices a and b indicating the beginning and end of the region to check

**Output**: The median of medians of subsets of size 7

for i from a to b, i incrementing by 7 //sorting in groups of 7 using 21 comparisons

for j from i to the lesser of (i+6) or b

for k from j+1 to lesser of (i+6) or b

if S[j] > S[k]

Swap S[j] and S[k]

x <- a

y <- a+3 //index of the first median

while y <= b //moving medians to the beginning

Swap S[x] and S[y]

x <- x+1

y <- y+7 //next median

for d from a to x-1 //sorting the medians, x-1 is the index of the last median

for e from d+1 to x-1

if S[d] > S[e]

Swap S[d] and S[e]

return (x-1-a)/2 + a //returns roughly the middle element of the sorted sequence of medians

**Algorithm** partition(a,b,S,p)

**Input**: An array, 𝑆, containing n elements; indices a and b indicating the beginning and end of the subarray; index p, indicating a previously chosen pivot

**Output**: The partitioned index of the previous selected pivot element

Swap S[p] and S[b]

l <- a

r <- b-1

while l <= r

while l <= r and S[l] <= S[b]

l <- l+1

while r >= l and S[r] >= S[b]

r <- r-1

if l < r

Swap S[l] and S[r]

Swap S[l] and S[b]

return l

* 1. {HHHH, HHHT, HHTH, HHTT, HTHH, HTHT, HTTH, HTTT, THHH, THHT, THTH, THTT, TTHH, TTHT, TTTH, TTTT}
  2. Event A, the event where heads are flipped first, occurs in 8 of 16 possible outcomes. Therefore, Pr(A) = 0.5.
  3. Event B, the event that there are exactly 2 heads and 2 tails flipped, occurs in 6 of 16 possible outcomes. Therefore, Pr(B) = 3/8 = 0.375.
  4. A∩B is the event where both A and B are true. We can count the occurrences in the sample space above to see that there are 3 in which both are true. Alternatively, we can calculate this using Pr(A) and Pr(B). Pr(A∩B) = 3/16 = Pr(A)\*Pr(B) = 0.5\*0.375 = 0.1875.
  5. E(X) = 0\*Pr(X = 0) + 1\*Pr(X = 1) + 2\*Pr(X = 2) + 3\*Pr(X = 3) + 4\*Pr(X = 4)

= 0\*1/16 + 1\*4/16 + 2\*6/16 + 3\*4/16 + 4\*1/16 = 2.

1. An algorithm to solve this would be a modified quicksort algorithm. As input, it takes an array of music files and the indices of the beginning and end of the array/subarray. Using a random file as a pivot, the algorithm partitions the array into unsorted subarrays, based on the comparisons made by Rustbucket. If an internal disk fault occurs, the comparison is retried. Files smaller than or equal to the pivot would go in the left subarray while files larger than the pivot will go into the right subarray. Once the partition is complete, the algorithm recursively runs on each of the subarrays.

Pick pivot: 1 time

Partition: Takes n-1 time without failures

50% chance of failure

Partition with 50% chance of failure will take 2(n-1) time

Chance to pick a good pivot (as defined in Lecture 4): 50%

We expect that 50% of the time, at least ¼ elements are split up or at most ¾ elements are in the same subarray

We expect the maximum average size of the smaller problem to be

50% bad pivot: n-1

50% good pivot: 3n/4

((n-1)+3n/4)/2 = n/2-0.5+3n/8 = 7n/8-0.5

In the expected case

T(n) = {1, n=1; T(7n/8-0.5)+ 2n -1, otherwise}

Show that T(n) <= cnlogn, given c > 0 and n > n0

T(7n/8-0.5) + 2n – 1 <= c(7n/8)log(7n/8) +2n -1 <= cnlogn

7clog(7n/8)/8 + 2 <= clogn + 1/n

7clog(7n/8)/8 <= clogn + 1/n

7clog(7n/8) <= 8clogn + 8/n

Since this is true, T(n) is O(nlogn)