1 Degenerate sorting

Given an input array of length $n$, where each element has a value of 1, describe the run times of the sorting algorithms covered in class (selectionSort, mergeSort, quickSort), using the $O(n)$ notation.

Selectionsort = $O(n^2)$
Mergesort = $O(n \log n)$

Grading: 3 points for each, max of 5 total.

2 Merge sort

Given an unsorted list of integers [15, 19, 0, 2, 1, 17, 100, 0, 12, 5, 18], show the list of intermediate arrays that will be produced during merge sort.

[15, 19, 0, 2, 1, 17, 100, 0, 12, 5, 18]

[15, 19] [0, 2, 1] [17, 100] [12, 5] [18]

[15] [19] [0] [2, 1] [17] [100] [12] [5] [18]

[2] [1] [100] [0] [5] [18]
3 Binary search

A recursive binary search runs on a sorted array $A$ of integers $[1, 4, 6, 8, 9, 12, 14, 15, 19, 20]$. We want to find key=17. List the sequence of elements of $A$ that will be compared with 17 during the search.

9, 15, 19

Grading: 4 points if used ceil consistently instead of floor, 5 points for perfect

4 Recurrences

Solve the following recurrences. Assume $T(n) \leq c$ for some constant $c$ and for all $n \leq 10$.

- $T(n) \leq 4T(n/2) + n$
- $T(n) \leq 4T(n/2) + n \log n$
- $T(n) \leq 2T(n/4) + \sqrt{n}$
- $T(n) \leq \sqrt{n}T(\sqrt{n}) + n$
- $T(n) \leq 7T(n/8) + n^{0.935784974}$

Solving the following recurrences. Assume $T(n) \leq c$ for some constant $c$ and for all $n \leq 10$.

Grading: 5 for perfectly correct. 3 points for generalizing recurrence for arbitrary $i$. 1 point for solving for $i$. 

2
• $T(n) \leq 4T(n/2) + n$

Solution:

$$T(n) \leq 4T\left(\frac{n}{2}\right) + n$$

$$\leq 4(4T\left(\frac{n}{4}\right) + \frac{n}{2}) + n = 4^2T\left(\frac{n}{4}\right) + 2n + n$$

$$\leq 4^2(4T\left(\frac{n}{8}\right) + \frac{n}{4}) + 2n + n = 4^3T\left(\frac{n}{8}\right) + 4n + 2n + n$$

$$\ldots$$

$$\leq 4^iT\left(\frac{n}{2^i}\right) + 2^{i-1}n + 2^{i-2}n + \ldots + 2n + n$$

$$= 4^iT\left(\frac{n}{2^i}\right) + \sum_{j=0}^{i-1} 2^jn$$

$$= 4^iT\left(\frac{n}{2^i}\right) + (2^i - 1)n$$

The basis case is $T(1) \leq c$, in which $n/2^i = 1$. We have $2^i = n$ and $i = \log n$. Therefore we have

$$T(n) \leq 4^iT\left(\frac{n}{2^i}\right) + (2^i - 1)n$$

$$= 2^{2i}T\left(\frac{n}{2^i}\right) + (2^i - 1)n$$

$$= (2^i)^2T\left(\frac{n}{2^i}\right) + (2^i - 1)n$$

$$= (n)^2T(1) + (n - 1)n$$

$$\leq cn^2 + (n - 1)n = O(n^2)$$

• $T(n) \leq 4T(n/2) + n \log n$

Solution:
\[ T(n) \leq 4T\left(\frac{n}{2}\right) + n \log n \]
\[ \leq 4\left(4T\left(\frac{n}{4}\right) + \frac{n}{2} \log \frac{n}{2}\right) + n \log n = 4^2T\left(\frac{n}{4}\right) + 2n \log \frac{n}{2} + n \log n \]
\[ \leq 4^2\left(4T\left(\frac{n}{8}\right) + \frac{n}{4} \log \frac{n}{4}\right) + 2n \log \frac{n}{2} + n \log n \]
\[ = 4^3T\left(\frac{n}{8}\right) + 4n \log \frac{n}{4} + 2n \log \frac{n}{2} + n \log n \]

\[ \cdots \]
\[ \leq 4^iT\left(\frac{n}{2^i}\right) + 2^{i-1}n \log \frac{n}{2^{i-1}} + 2^{i-2}n \log \frac{n}{2^{i-2}} + \cdots + 2n \log \frac{n}{2} + n \log n \]
\[ = 4^iT\left(\frac{n}{2^i}\right) + \sum_{j=0}^{i-1} 2^j n \log \frac{n}{2^j} \]
\[ = 4^iT\left(\frac{n}{2^i}\right) + \sum_{j=0}^{i-1} 2^j n (\log n - \log 2^j) \]
\[ = 4^iT\left(\frac{n}{2^i}\right) + \sum_{j=0}^{i-1} 2^j n \log n - \sum_{j=0}^{i-1} 2^j jn \]
\[ = 4^iT\left(\frac{n}{2^i}\right) + (2^i - 1) n \log n - \sum_{j=0}^{i-1} 2^j jn \]
\[ = 4^iT\left(\frac{n}{2^i}\right) + (2^i - 1) n \log n - (2 - 1) \cdot \sum_{j=0}^{i-1} 2^j jn \]
\[ = 4^iT\left(\frac{n}{2^i}\right) + (2^i - 1) n \log n - 2 \sum_{j=0}^{i-1} 2^j jn + \sum_{j=0}^{i-1} 2^j jn \]
\[ = 4^iT\left(\frac{n}{2^i}\right) + (2^i - 1) n \log n - \sum_{j=0}^{i-1} 2^{j+1} jn + \sum_{j=0}^{i-1} 2^j jn \]
\[ = 4^iT\left(\frac{n}{2^i}\right) + (2^i - 1) n \log n - n\left(\sum_{j=0}^{i-1} 2^{j+1} j - \sum_{j=0}^{i-1} 2^j j\right) \]
\[ = 4^iT\left(\frac{n}{2^i}\right) + (2^i - 1) n \log n - n\left(\sum_{k=1}^{i} 2^k (k - 1) - \sum_{j=0}^{i-1} 2^j j\right) \]
\[ = 4^iT\left(\frac{n}{2^i}\right) + (2^i - 1) n \log n - n\left(\sum_{k=1}^{i} 2^k - \sum_{k=1}^{i} 2^k - \sum_{j=0}^{i-1} 2^j j\right) \]
\[ = 4^iT\left(\frac{n}{2^i}\right) + (2^i - 1) n \log n - n\left(\sum_{k=0}^{i-1} 2^k k + 2^i i - \sum_{k=1}^{i} 2^k - \sum_{j=0}^{i-1} 2^j j\right) \]
\[ = 4^iT\left(\frac{n}{2^i}\right) + (2^i - 1) n \log n \left(\frac{1}{4} n(2^i - \sum_{k=1}^{i} 2^k) \right) \]
\[ = 4^iT\left(\frac{n}{2^i}\right) + (2^i - 1) n \log n - n(2^i (2^i - (2^{i+1} - 2)) \right) \]
\[ = 4^iT\left(\frac{n}{2^i}\right) + (2^i - 1) n \log n - 2^i n + 2^{i+1} n - 2n \]
The basis case is \( T(1) \leq c \), in which \( n/2^i = 1 \). We have \( 2^i = n \) and \( i = \log n \). Therefore we have

\[
T(n) \leq 4^i T\left( \frac{n}{2^i} \right) + (2^i - 1)n \log n - 2^i n \log n - 2^{i+1} n - 2n
\]

\[
= (2^i)^2 T\left( \frac{n}{2^i} \right) + (2^i - 1)n \log n - 2^i n \log n + 2 \cdot 2^i n - 2n
\]

\[
= (n)^i T(1) + (n - 1)n \log n - n^2 \log n + 2n^2 - 2n
\]

\[
\leq cn^2 + (n - 1)n \log n - n^2 \log n + 2n^2 - 2n
\]

\[
= (c + 2)n^2 - n \log n - 2n
\]

\[
= O(n^2)
\]

- \( T(n) \leq 2T(n/4) + \sqrt{n} \)

Solution:

\[
T(n) \leq 2T\left( \frac{n}{4} \right) + \sqrt{n}
\]

\[
\leq 2(2T\left( \frac{n}{4^2} \right) + (\frac{n}{4})^{\frac{1}{2}}) + n^{\frac{1}{2}} = 2^2 T\left( \frac{n}{4^2} \right) + 2n^{\frac{1}{2}}
\]

\[
\leq 2^2 (2T\left( \frac{n}{4^3} \right) + (\frac{n}{16})^{\frac{1}{2}}) + 2n^{\frac{1}{2}} = 2^3 T\left( \frac{n}{4^3} \right) + 3n^{\frac{1}{2}}
\]

\[
\cdots
\]

\[
\leq 2^iT\left( \frac{n}{4^i} \right) + in^{\frac{1}{2}}
\]

The basis case is \( T(1) \leq c \), in which \( n/4^i = 1 \). We have \( 4^i = n \) and \( i = \log_4 n \). Therefore we have

\[
T(n) \leq 2^iT\left( \frac{n}{4^i} \right) + in^{\frac{1}{2}}
\]

\[
\leq c2^{\log_4 n} + \sqrt{n} \log_4 n
\]

\[
= c2^{\frac{1}{2}\log n} + \sqrt{n} (\frac{1}{2} \log n)
\]

\[
= c(2^{\log n})^{\frac{1}{2}} + \frac{1}{2} \sqrt{n} \log n
\]

\[
= c\sqrt{n} + \frac{1}{2} \sqrt{n} \log n
\]

\[
= O(\sqrt{n} \log n)
\]

- \( T(n) \leq \sqrt{n} T(\sqrt{n}) + n \)

5
Solution:

Let $n = 2^k$, we solve for the recurrence of $T(2^k)$:

\[
T(2^k) \leq 2^k T(2^k/2) + 2^k \\
\leq 2^k (2^k T(2^k/2) + 2^k) + 2^k = 2^{(k+1)/2} T(2^k/2) + 2 \cdot 2^k \\
\leq 2^{(k+1)/2} T(2^k/2) + 2 \cdot 2^k \\
= 2^{(k+1)/2} T(2^k/2) + 3 \cdot 2^k \\
\vdots \\
\leq 2^{(k+1)/2} T(2^k/2) + i \cdot 2^k \\
= 2^{(1-\frac{k}{2^i})} T(2^k/2) + i \cdot 2^k
\]

The basis case is $T(2) \leq c$, in which $2^{\frac{k}{2}} = 2$. We have $k = 2^i$ and $i = \log k$. Therefore we have

\[
T(2^k) \leq 2^{(1-\frac{k}{2^i})} T(2^k/2) + i \cdot 2^k \\
= 2^0 T(2) + 2^k \log k \\
\leq c + 2^k \log k
\]

and since $k = \log n$, we have

\[
T(n) \leq c + n \log \log n = O(n \log \log n)
\]

- $T(n) \leq 7T(n/8) + n^{0.935784974}$

Solution:
Let $t = 0.935784974$, we solve for the recurrence of $T(n)$:

$$ T(n) \leq 7T\left(\frac{n}{8}\right) + n^t $$

$$ \leq 7(7T\left(\frac{n}{8^2}\right) + \left(\frac{n}{8}\right)^t + n^t $$

$$ = 7^2T\left(\frac{n}{8^2}\right) + \frac{7}{8^t}n^t + n^t $$

$$ \leq 7^2(7T\left(\frac{n}{8^3}\right) + \left(\frac{n}{8^2}\right)^t + \frac{7}{8^t}n^t + n^t $$

$$ = 7^3T\left(\frac{n}{8^3}\right) + \left(\frac{7}{8^2}\right)^2n^t + \left(\frac{7}{8^t}\right)n^t + n^t $$

$$ \ldots $$

$$ \leq 7^iT\left(\frac{n}{8^i}\right) + \left(\frac{7}{8^i}\right)^{i-1} + \left(\frac{7}{8^i}\right)^{i-2} + \ldots + \left(\frac{7}{8^i}\right) + 1)n^t $$

$$ = 7^iT\left(\frac{n}{8^i}\right) + \left(1 - (\frac{7}{8^i})\right) n^t $$

since $t = 0.935784974$ is a constant, we denote the constant term $1/(1 - \frac{7}{8^t}) = c_1$, the recurrence becomes

$$ T(n) \leq 7^iT\left(\frac{n}{8^i}\right) + c_1(1 - (\frac{7}{8^i})n^t $$

$$ = 7^iT\left(\frac{n}{8^i}\right) + c_1n^t - c_1\left(\frac{7}{8^i}\right)n^t $$

The basis case is $T(1) \leq c$, in which $n/8^i = 1$. We have $n = 8^i$ and $i = \log_8 n$. Therefore we have

$$ T(n) \leq 7^iT\left(\frac{n}{8^i}\right) + c_1n^t - c_1\left(\frac{7}{8^i}\right)n^t $$

$$ = 7^{\log_8 n}T(1) + c_1n^t - c_1\left(\frac{7}{8^i}\right)n^t $$

$$ \leq c_1^{\log_8 n}n + c_1n^t - c_1\left(\frac{7}{8^i}\right)n^t $$

$$ = c_1^{\log_8 n}n + c_1n^t - c_1\left(\frac{\log_8 n}{n^t}\right)n^t $$

$$ = c_1^{\log_8 n}n + c_1n^t - c_1\left(\frac{\log_8 n}{n^t}\right) $$

$$ = (c - c_1)^{\log_8 n} + c_1n^t $$

7
since $7^{\log_8 n} = (n^{\log_8 7})^{\log_8 n} = n^{(\log_8 7) \cdot (\log_8 n)} = n^{\log_8 7}$, we have

$$T(n) \leq (c - c_1)n^{\log_8 7} + c_1 n^t$$

and we know that $\log_8 7 = 0.93578497401 > t$, therefore

$$T(n) = O(n^{\log_8 7})$$

5 Partition

The following array has been partitioned. Which elements could have been the pivot value?

$$[31, 0, 25, 47, 53, 82, 79, 64, 98]$$

(Moved to next assignment, but, hey, here’s the solution.)

Solution:

In a partition operation, all elements with values less than the pivot locate at one side of the pivot while all elements greater than the pivot locate at another side. In the array, 47, 53, 98 could have been the pivot.

Grading: deferred to next homework.

6 Duplicates

Write an algorithm that, given an array $A$, outputs an array $B$ that has the same set of elements as $A$, but does not have any duplicate elements. $B$ need not list the elements of $A$ in the same order they occur in $A$. What is the running time of your algorithm.

Now suppose you want $B$ to have the same order as the first occurrence of each element in $A$. Write an algorithm and give its running time.

Part 1: Use your favorite sorting algorithm (i.e. merge sort), to sort array $A$. Go through the sorted elements of array $A$, in order, and output each item only if it is not equal to the previous one. Runtime is $O(n \log n) + O(n) = O(n \log n)$.

Part 2: Use part 1 to get a sorted array of unique elements, $C$. Also, create a bit vector, and set it to all 0s. Take the unsorted array, $A$, and for each element, do a binary search on the sorted array of unique elements, $C$. If you find the element, check the bit in the
bit vector, and if it is 0, set it to 1, and output the element; if the bit is already set, do not output the element. Runtime is \( O(n \log n) + O(n \log n) + O(n) = O(n \log n) \)

Grading:

- 10 points for perfect
- 5 points for some understanding, 10 points for perfect

7 2D searching

You are given a two-dimensional matrix \( A \) of size \( m \times n \). Each row of \( A \) is sorted. Each column of \( A \) is sorted. Analyze the running time of the following algorithms for searching for an element \( t \) in \( A \).

- Check each element of \( A \) to see if it is equal to \( x \).
- Perform binary search for \( x \) in each row of \( A \).
- Perform binary search for \( x \) in each column of \( A \).
- \( \text{recursiveSearch1}(A, xlo, xhi, ylo, yhi, t) \)
  - if \( xhi < xlo \) or \( yhi < ylo \), then return NOTFOUND
  - let \( xmid = \lfloor (xhi - xlo) / 2 \rfloor \)
  - let \( ymid = \lfloor (yhi - ylo) / 2 \rfloor \)
  - if \( A[xmid][ymid] == t \) then return \( (xmid, ymid) \)
  - else if \( A[xmid][ymid] < t \)
    * \( a = \text{recursiveSearch1}(A, xmid + 1, xhi, ylo, ymid, t) \)
    * \( b = \text{recursiveSearch1}(A, xlo, xmid, ymid + 1, yhi, t) \)
    * \( c = \text{recursiveSearch1}(A, xmid + 1, xhi, ymid + 1, yhi, t) \)
  - else
    * \( a = \text{recursiveSearch1}(A, xmid, xhi, ylo, ymid - 1, t) \)
    * \( b = \text{recursiveSearch1}(A, xlo, xmid - 1, ymid, yhi, t) \)
    * \( c = \text{recursiveSearch1}(A, xlo, xmid - 1, ylo, ymid - 1, t) \)
  - if any of \( a, b, \) or \( c \) is not NOTFOUND, return it
  - else return NOTFOUND
• Define and analyze an algorithm similar to the above, except that instead of picking the midpoint of $A$, it performs binary search on the diagonal of $A$ and recurses based on that.

• Check each element of $A$ to see if it is equal to $x$ 
$O(mn)$ (5 points)

• Perform binary search for $x$ in each row of $A$ 
$O(m \log n)$ (5 points, 3 if reversed $m$ and $n$)

• Perform binary search for $x$ in each column of $A$ 
$O(n \log m)$ (5 points, 3 if reversed $m$ and $n$)

• recursiveSearch1($A$, $xlo$, $xhi$, $ylo$, $yhi$, $t$)

**Solution:** Let $N = m \times n$. The recurrence is:

$$T(N) \leq 3T\left(\frac{N}{4}\right) + 1$$

$$\leq 3(3T\left(\frac{N}{4^2}\right) + 1) + 1$$

$$= 3^2T\left(\frac{N}{4^2}\right) + 3 + 1$$

$$\leq 3^2(3T\left(\frac{N}{4^3}\right) + 1) + 3 + 1$$

$$= 3^3T\left(\frac{N}{4^3}\right) + 3^2 + 3 + 1$$

$$\vdots$$

$$\leq 3^iT\left(\frac{N}{4^i}\right) + 3^{i-1} + 3^{i-2} + \ldots + 3^2 + 3 + 1$$

$$= 3^iT\left(\frac{N}{4^i}\right) + \sum_{j=0}^{i-1} 3^j$$

$$= 3^iT\left(\frac{N}{4^i}\right) + \frac{1}{2}(3^i - 1)$$

The basis case is $T(1) = 1$, in which $N/4^i = 1$. We have $4^i = N$ and $i = \log_4 N$. 

10
Therefore we have

\[ T(n) \leq 3^iT\left(\frac{N}{4}\right) + \frac{1}{2}(3^i - 1) \]

\[ = 3^{\log_4 N}T(1) + \frac{1}{2}(3^{\log_4 N} - 1) \]

\[ = 3^{\log_4 N} + \frac{1}{2}(3^{\log_4 N} - 1) \]

\[ = \frac{3}{2}3^{\log_4 N} - \frac{1}{2} \]

\[ = \frac{3}{2}3^{\log_4 N} - \frac{1}{2} \]

\[ = \frac{3}{2}^{\log_4 3} - \frac{1}{2} \]

\[ = O(N^{\log_4 3}) \]

Grading: 10 points total. 5 points for writing down the correct recurrence. Solving the recurrence graded just like problem 4.

- Define and analyze an algorithm similar to the above, except that instead of picking the midpoint of A, it performs binary search on the diagonal of A and recurses based on that.

Grading: This one was difficult, so we just gave 10 points for any complete and correct algorithm. No analysis needed.

**Solution 1:**

From the definition we know that

\[ A_{i,j} \leq A_{m,n}, \forall i \leq m, \forall j \leq n \]

The main diagonal of A is a sorted 1-dimensional array.

Let \((F, P) = \text{binarySearch}(X, x)\) denotes the binary search algorithm for find element x in a 1-D array X. If \(F = \text{True}\), then the element is found while if \(F = \text{False}\), the element is not found. \(P\) is the last searching position.

The following algorithm performs recursive binary search on the main diagonal of matrix A to find element t, in which F indicates if the element is found and P2 returns the final searching position. Let \(a = 0, b = 0, c = (m - 1), d = (n - 1)\):

\((F, P2) = \text{recursiveSearch2}(A, a, b, c, d, t)\)
• if $a > c$ or $b > d$, then return $F = \textbf{False}$.

• Let $B_{(c-a) \times (d-b)}$ be the submatrix of matrix $A$ such that $B_{i,j} = A_{(a+i),(b+j)}$. Let $\text{diag}B$ be the main diagonal of $B$ such that $\text{diag}B$ is the shortest path between the top-left corner($B_{0,0}$) and bottom-right corner($B_{(c-a-1),(d-b-1)}$) and $\text{diag}B$ separate the matrix into 2 equal parts.

• $(F,P) = \text{binarySearch}(\text{diag}B,t)$.

• If $F = \textbf{True}$, return $F$ and $P2 = (a + P, b + P)$.

• Else $(F,P2) = \text{recursiveSearch2}(A,a,(b + P),(a + P),d,t)$.

• If $F = \textbf{True}$, return $F$ and $P2$.

• Else $(F,P2) = \text{recursiveSearch2}(A,(a + P),b,c,(b + P),t)$.

• Return $F$ and $P2$.

Note that for a given matrix $B$ of size $m \times n$, the length of the diagonal $\text{diag}B$ is $L = \max(m,n)$. $\forall(0 < i < L)$, if $\text{diag}B_i = B_{p,q}$, then $\text{diag}B_{i+1} \in \{B_{p,q+1}, B_{p+1,q}, B_{p+1,q+1}\}$.

Assuming $N = m \times n$ and the complexity of finding the diagonal of a $m \times n$ matrix $f(m,n) = c$ is a constant, the recurrence is

$$T(N) \leq 2T\left(\frac{N}{4}\right) + \log N + c$$

$$\leq 2(2T\left(\frac{N}{4^2}\right) + \log \frac{N}{4}) + \log N + 2c + c$$

$$\cdots$$

$$\leq 2^i T\left(\frac{N}{4^i}\right) + \sum_{j=0}^{i-1} 2^j (\log \frac{N}{4^j} + c)$$

$$= 2^i T\left(\frac{N}{4^i}\right) + (2^i - 1) \log N - 2(i2^i - 2^{i+1} + 2) + (2^i - 1)c$$

The basis case is $T(1) = 1$ , in which $N/4^i = 1$. We have $4^i = N$, $2^i = \sqrt{N}$ and $i = \log_4 N$. Therefore we have

$$T(N) \leq 2^i T\left(\frac{N}{4^i}\right) + (2^i - 1) \log N - 2(i2^i - 2^{i+1} + 2) + (2^i - 1)c$$

$$= \sqrt{N} + (\sqrt{N} - 1) \log N - 2(\sqrt{N} \log_4 N - 2\sqrt{N} + 2) + (\sqrt{N} - 1)c$$

$$= O(\sqrt{N})$$
Solution 2:

From the definition we know that

\[ A_{i,j} \leq A_{m,n}, \forall i \leq m, \forall j \leq n \]

The main diagonal of A is a sorted 1-dimensional array.

Let \((F, P) = \text{binarySearch}(X, x)\) denotes the binary search algorithm for find element \(x\) in a 1-D array \(X\). If \(F = \text{True}\), then the element is found while if \(F = \text{False}\), the element is not found. \(P\) is the last searching position.

The following algorithm performs recursive binary search on the main diagonal of matrix \(A\) to find element \(t\), in which \(F\) indicates if the element is found and \(P\) returns the final searching position. Let \(a = 0, b = 0, c = (m - 1), d = (n - 1)\):

\((F, P2) = \text{recursiveSearch2}(A, a, b, c, d, t)\)

- If \(a > c\) or \(b > d\), then return \(F = \text{False}\).
- Let \(B_{(c-a)\times(d-b)}\) be the submatrix of matrix \(A\) such that \(B_{i,j} = A_{(a+i),(b+j)}\). Let \(\text{diag}B\) be the main diagonal of \(B\) such that \(\text{diag}B_i = B_{i,i}\).

\((F, P) = \text{binarySearch}(\text{diag}B, t)\).
- If \(F = \text{True}\), return \(F\) and \(P2 = (a + P, b + P)\).
- Else \((F, P2) = \text{recursiveSearch2}(A, a, (b + P), (a + P), d, t)\).
- If \(F = \text{True}\), return \(F\) and \(P2\).
- Else \((F, P2) = \text{recursiveSearch2}(A, (a + P), b, c, (b + P), t)\).
- Return \(F\) and \(P2\).

Note that for a given matrix \(B\) of size \(m \times n\), the length of the diagonal \(\text{diag}B\) is \(\min(m, n)\), the recurrence is

\[ T(m, n) = T(m_1, n_1) + T(m_2, n_2) + \log(\min(m, n)) \]

such that \(m = m_1 + m_2\) and \(n = n_1 + n_2\).

(Proof that this is still efficient forthcoming.)