CS 473 ✧ Fall 2024
Midterm 1 Problem 1 Solution

Prove that every integer (positive, negative, or zero) can be written in the form $\sum_i \pm 3^i$, where the exponents i are distinct non-negative integers.

Solution (direct induction): This notation is called ***balanced ternary***.

Let n be an arbitrary integer. Assume that any non-negative integer m such that $|m| < |n|$ can be written in balanced ternary. There are four cases to consider.

- The base case $n = 0$ is trivial—zero is the empty sum.
- Suppose $n = 3m$ for some integer $m \neq 0$. Because $|m| < |n|$, the inductive hypothesis implies that m can be written in balanced ternary. Shifting all exponents up by 1 gives us a balanced ternary representation of n.
- Suppose $n = 3m + 1$ for some integer m. Because $|m| < |n|$, the inductive hypothesis implies that m can be written in balanced ternary. Shifting all exponents up by 1 and adding 3^0 gives us a balanced ternary representation of n.
- Finally, suppose $n = 3m - 1$ for some integer m. Because $|m| < |n|$, the inductive hypothesis implies that m can be written in balanced ternary. Shifting all exponents up by 1 and subtracting 3^0 gives us a balanced ternary representation of n.

In all cases, we conclude that n can be written in balanced ternary. ■

Solution (induction and symmetry): This notation is called ***balanced ternary***.

Let n be an arbitrary integer. If n can be written in balanced ternary, then obviously so can $-n$, by inverting the sign of every term (or equivalently, by negating every *trit*). Thus, without loss of generality, we can assume that n is non-negative.

We complete the proof by induction on n. Assume that any non-negative integer $m < n$ can be written in balanced ternary. There are two cases to consider.

- The base case $n = 0$ is trivial—zero is the empty sum.
- Otherwise, let $m = \lfloor (n+1)/3 \rfloor$; this is the integer closest to $n/3$. Because $m < n$, the inductive hypothesis implies that m can be written in balanced ternary. Shifting all exponents up by 1 gives us a balanced ternary representation of $3m$ that does not include 3^0. There are three subcases to consider:
  - If $n = 3m$, we're done.
  - If $n = 3m + 1$, adding 3^0 gives us a balanced ternary representation of n.
  - If $n = 3m - 1$, subtracting 3^0 gives us a balanced ternary representation of n.

In all cases, we conclude that n can be written in balanced ternary. ■

Rubric: 10 points = 2 for valid strong inductive hypothesis + 2 for explicit exhaustive case analysis + 1 for base case(s) + 2 for correctly applying the stated induction hypothesis + 3 for other details of the inductive case(s). These are not the only correct proofs.

A proof that only considers only non-zero integers is worth at most 9 points; only non-negative integers is worth at most 7; only positive integers is worth at 6 points.

CS 473 ✧ Fall 2024
Midterm 1 Problem 2 Solution

Describe and analyze an algorithm to find the maximum number of points you can earn in a Number Blast puzzle. (See the question handout for a detailed description of the rules.) The input to your algorithm is an array $A[1..2n]$ of positive integers.

Solution (just like HW3.2): For any indices i and j such that $j-i$ is odd, let $\mathit{MaxBlast}(i,j)$ denote the maximum score we can earn using only squares in the interval $A[i..j]$. Considering all possible *last* moves (i',j') gives us the following recurrence:

$$\mathit{MaxBlast}(i,j) = \begin{cases} -\infty & \text{if } j-i \text{ is even} \\ 0 & \text{if } i = j+1 \\ \max \left\{ \begin{array}{l} A[i'] \cdot A[j'] \\ \quad + \mathit{MaxBlast}(i, i'-1) \\ \quad + \mathit{MaxBlast}(i'+1, j'-1) \\ \quad + \mathit{MaxBlast}(j'+1, j) \end{array} \;\middle|\; i \le i' < j' \le j \right\} & \text{otherwise} \end{cases}$$

We can memoize this function into a two-dimensional array. We can evaluate the array using two nested for-loops, one decreasing i and the other increasing j. (It doesn't matter how these loops are nested.) The resulting algorithm runs in ***$O(n^4)$ time***. ■

Solution (leftmost turn, 12/10): For any indices i and k such that $k-i$ is odd, let $\mathit{MaxBlast}(i,k)$ denote the maximum score we can get earn using only squares in the interval $A[i..k]$. At some point during the puzzle, we must choose the leftmost square $A[i]$ along with another square $A[j]$ with $i < j \le k$. Without loss of generality, this is the last move of the puzzle! Considering all possible indices j gives us the following recurrence:

$$\mathit{MaxBlast}(i,k) = \begin{cases} -\infty & \text{if } k-i \text{ is even} \\ 0 & \text{if } i = k+1 \\ \max \left\{ \begin{array}{l} A[i] \cdot A[j] \\ \quad + \mathit{MaxBlast}(i+1, j-1) \\ \quad + \mathit{MaxBlast}(j+1, k) \end{array} \;\middle|\; i < j \le k \right\} & \text{otherwise} \end{cases}$$

We can memoize this function into a two-dimensional array. We can evaluate the array using two nested for-loops, one decreasing i and the other increasing k. (It doesn't matter which one is the inner loop.) The resulting algorithm runs in ***$O(n^3)$ time***. ■

Rubric: 10 points; standard dynamic programming rubric. These are not the only correct solutions. These solution includes more justification than necessary for full credit. An algorithm that runs in $O(n^4)$ time is worth full credit. Max 12 points (yes, out of 10) for $O(n^3)$ time; max 8 points for $O(n^5)$ time; scale extra/partial credit.

CS 473 ✧ Fall 2024
Midterm 1 Problem 3 Solution

(a) Describe an algorithm to determine the number of well-spaced triples in a given bit string $B[1..n]$.

(b) Describe an algorithm to determine the number of offset triples in a given bit string $B[1..n]$.

Solution (part (a)): The following algorithm runs in ***$O(n \log n)$ time***; the running time is dominated by the initial convolution.

```
CountEvenTriples(B[0..n]):
  BB ← Convolution(B, B)
  triples ← 0
  for j ← 0 to n
      if B[j] = 1
          triples ← triples + ⌊BB[2j]/2⌋
  return triples
```

Justification: In any well-spaced triple $\{i, j, k\}$, the middle index j is the average of the other two indices i and k, and without loss of generality $i < j < k$. Thus, if $B[j] = 1$, the number of well-spaced triples with middle index j is equal to

$$\sum_{\substack{i+k=2j \\ i<j<k}} B[i] \cdot B[k].$$

This is *almost* equal to the $(2j)$th element of the convolution $B * B$:

$$(B * B)[2j] = \sum_{i+k=2j} B[i] \cdot B[k].$$

The latter sum actually counts well-spaced triples twice, once as $\{i, j, k\}$ and again as $\{k, j, i\}$; it also counts the degenerate triple $\{j, j, j\}$. So in fact, j is the middle index of c evenly-spaced triples if and only if $(B * B)[2j] = 2c + 1$. ■

Rubric: 5 points = 2 for using FFT/convolution + 2 for correctly extracting the number of triples from the convolution + 1 for time analysis. The justification is *not* required for full credit. This is not the only correct solution.

Solution (part (b)): The following algorithm runs in $O(n \log n)$ *time*; the running time is dominated by the initial convolution.

```
CountOffTriples(B[0..n]):
  for i ← 0 to n
      BB[2i] ← B[i]
      BB[2i+1] ← 0
  BBB ← Convolution(BB, B)
  triples ← 0
  for j ← 0 to n
      if B[j] = 1
          triples ← triples + BBB[3j] − 1
  return triples
```

Justification: The indices of offset triple $\{i, j, k\}$ satisfy the equation $3j = 2i + k$. Thus, if $B[j] = 1$, the number of offset triples with middle index j is equal to

$$\sum_{\substack{2i+k=3j\\ i\neq j\neq k}} B[i]\cdot B[k].$$

This is again *almost* equal one element of a convolution. Let BB be a copy of B interleaved with 0's, as shown in the pseudocode above. Then we have

$$\begin{aligned}(BB * B)[3j] &= \sum_{i'+k=3j} BB[i']\cdot B[k]\\ &= \sum_{2i+k=3j} BB[2i]\cdot B[k] && [i' = 2i]\\ &= \sum_{2i+k=3j} B[i]\cdot B[k]\end{aligned}$$

This sum includes the degenerate triple $\{j, j, j\}$, so we have to subtract 1 to get the actual number of offset triples with middle index j. ■

Rubric: 5 points = 1 for using FFT/convolution + 2 for correctly setting up the convolution + 1 for correctly computing the number of offset triples from the convolution + 1 for time analysis. The justification is *not* required for full credit. This is not the only correct solution.

CS 473 ✧ Fall 2024
Midterm 1 Problem 4 Solution

Describe an algorithm that finds the smallest possible total badness for a given sequence of words. The input to your algorithm consists of the positive integer M (the number of characters that fit on a single line) and an array $L[1..n]$, where $L[i]$ is the length of the ith word.

Solution (word-by-word): Let $MinBad(i, L)$ denote the smallest possible total badness for words i through n, assuming we have room for L characters on the first line, and M characters on every later line. We need to compute $MinBad(1, M)$. This function satisfies the following recurrence:[a]

$$MinBad(i,L) = \begin{cases} badness(L) & \text{if } i > n \\ MinBad(i+1,\ M - L[i]) & \text{if } L = M \\ badness(L) + MinBad(i+1,\ M - L[i]) & \text{if } L[i] > L - 1 \\ \min \left\{ \begin{array}{c} MinBad(i+1,\ L - L[i] - 1) \\ badness(L) + MinBad(i+1,\ M - L[i]) \end{array} \right\} & \text{otherwise} \end{cases}$$

The choices in the final recursive case correspond to putting a space followed by the ith word on the current line, or starting a new line with the ith word. The -1s in red account for the spaces between words.

We can memoize this function into a two-dimensional array $MinBad[1..n, 0..M]$. We can evaluate this array using two nested loops, decreasing i in the outer loop and considering L in any order in the inner loop. The resulting algorithm runs in ***$O(nM)$ time***. ■

[a]This recurrence assumes that the paragraph layout must use at least one line. This assumption is trivial if n is positive, but when $n = 0$, the correct value of $MinBad(1, M)$ is arguably 0, not $badness(M)$.

Solution (line-by-line, 11/10): Let $PrefLen(j) = \sum_{i=1}^{j} L[i]$ denote the sum of the first j word lengths. We can compute all values of this function in $O(n)$ time as follows:

```
PrefLen[0] ← 0
for i ← 1 to n
    PrefLen[i] ← PrefLen[i − 1] + L[i]
```

Now let $LineLen(i, j)$ denote the length of a single line containing words $i, i+1, \ldots, j$. We immediately have

$$LineLen(i, j) = PrefLen(j) - PrefLen(i-1) + j - i$$

(The $j - i$ term at the end counts the spaces between words.)

Finally, let $MinBad(i)$ denote the smallest possible total badness for words i through n, assuming every line has length M. We need to compute $MinBad(1)$. This function satisfies the following recurrence, which tries all possibilities for the first line:

$$MinBad(i) = \begin{cases} 0 & \text{if } i > n \\ \min \left\{ \begin{array}{c} badness(M - LineLen(i,j)) \\ + MinBad(j+1) \end{array} \;\middle|\; \begin{array}{c} i \le j \le n \quad \text{and} \\ LineLen(i,j) \le M \end{array} \right\} & \text{otherwise} \end{cases}$$

We can memoize this function into an array $MinBad[1..n]$, which we can fill in decreasing index order. Because each word length is positive (or alternatively, thanks to the spaces between words), each entry $MinBad[i]$ depends on at most n *and* at most M later entries $MinBad[j]$. Thus, the resulting algorithm runs in $\boldsymbol{O(\min\{n^2, nM\})}$ ***time***. ■

Rubric: 10 points: standard dynamic programming rubric. The explanations in gray are not required for full credit. −1 for ignoring spaces between words. +1 extra credit for $O(\min\{n^2, nM\})$ time. (In practice, $n > L$ seems more likely than $n < L$, so this is a small improvement.) These are not the only correct solutions.