• **Homework 1**
  • Due Friday at 2pm
• **Midterm**
  • Tuesday, Mar 8
  • In-class contest
Exercise  Given an integer $x$, write a function $\text{reverseBytes}(x)$ that returns an integer whose bytes are reversed. For example, $\text{reverseBytes}(3628) = 739115008$ because $3628 = 0x00000E2C$ and $739115008 = 0x2C0E0000$. 
reverseBytes(x)
return (x & 0x000000FF) << 24
   | (x & 0x0000FF00) << 8
   | (x & 0x00FF0000) >> 8
   | (x & 0xFF000000) >>> 24
...or Integer.reverseBytes(x) in Java
Exercise  Write the following functions using bitmasks: `isEven(x)`, `evenBitsOnly(x)`, `setNthBit(x, N)`. For example:

- `isEven(5)` returns `false`
- `evenBitsOnly(14)` returns `10`
- `setNthBit(11, 2)` returns `15`
• $2^{10} = 1024 \approx 1K$
• $2^{20} = 1048576 \approx 1M$
• $10! = 3628800 \neq 3M$
• Max signed integer value: $2^{31} - 1 = 2147483647 \approx 2B$
• Power set of a set of size N: $2^N$
• If all you care about is sum range queries, then you can use a Fenwick (binary indexed) tree
  • On a list that may be dynamically updated (values are increased/decreased)
• Short and efficient implementation
  • Implemented as an array/vector
• Determines sum of numbers in range [1, k] in $O(\log n)$ time
  • **Warning**: following code uses 1-based indices for cleaner code
  • For an arbitrary range, find two ranges and subtract
• Suppose we want to find the sum of elements of list L in the range \([1, k]\)
  • Remember: 1-based index
• \(k = 2^{e_1} + 2^{e_2} + \ldots + 2^{e_n}\) where \(e_1 < \ldots < e_n\)

\[
\sum_{i=1}^{k} L[i] = \sum_{i=1}^{k-2^{e_1}} L[i] + \sum_{i=k-2^{e_1}+1}^{k} L[i]
\]

• Rightmost sum stored in our table at index \(k\)
• Leftmost sum can be evaluated recursively
• Suppose \( k = 26 = 110102 \), what would the Fenwick tree table entry \( F[k] \) be? What if \( k = 24 \)?
public int sumQuery(int a, int b) {
    return sumQuery(b) - sumQuery(a - 1);
}

public int sumQuery(int k) {
    int ret = 0;
    while (k > 0) {
        ret += table[k];
        k &= k - 1;
    }
    return ret;
}

public void adjust(int i, int adj) {
    while (i < table.length) {
        table[i] += adj;
        i += (i & (-i));
    }
}
• Build a Fenwick tree (table) for L = [1, 2, 1, 3, -1, 1]
• Find the sum of the range [3, 6]

<table>
<thead>
<tr>
<th>Index_{10}</th>
<th>Index_{2}</th>
<th>Interval</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>[1, 1]</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>[1, 2]</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>[3, 3]</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>[1, 4]</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>[5, 5]</td>
<td>-1</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>[5, 6]</td>
<td>0</td>
</tr>
</tbody>
</table>
• What happens to the table when L[1] is updated from 1 to 4?
  • All entries corresponding to ranges including affected index need to be updated

<table>
<thead>
<tr>
<th>Index_{10}</th>
<th>Index_2</th>
<th>Interval</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>[1, 1]</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>[1, 2]</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>[3, 3]</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>[1, 4]</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>[5, 5]</td>
<td>−1</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>[5, 6]</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index_{10}</th>
<th>Index_2</th>
<th>Interval</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>[1, 1]</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>[1, 2]</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>[3, 3]</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>[1, 4]</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>[5, 5]</td>
<td>−1</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>[5, 6]</td>
<td>0</td>
</tr>
</tbody>
</table>
• If you don’t store the original list but have its Fenwick tree, how can you rebuild the original list?
• Can you use a Fenwick tree for min/max queries?
Suppose you have a set of \( n = 10^6 \) of the numbers \( \{1, 2, \ldots, n\} \). You are given a list of instructions each taking one of the following forms:

(a) Remove the number \( k \).

(b) Output the \( j \)th largest remaining number.

Explain how to use a Fenwick tree to deal with each instruction in \( \log(n) \) time.
Suppose you are given a list of instructions regarding the multiset $S$ each taking one of the following forms:

(a) Insert the value $k$, where $k$ is a 32-bit signed integer.
(b) Remove the value $k$, where $k$ is a 32-bit signed integer.
(c) Give the percentage of the list smaller than $x$, where $x$ is a 32-bit signed integer.
(d) Give the $j$th largest element in the set.

Assume further that you are guaranteed only $10^6$ distinct values will occur (but each may occur many times). Explain how to a Fenwick tree to efficiently handle these instructions.
Competitive Programming 2.4.4