| | | |
|---|---|---|
| | **6.854 Advanced Algorithms** | |
| Lecture 4: September 15, 2003 | | Lecturer: Erik Demaine |
| | Scribes: Joy M. Forsythe | |

**Integer Sorting**

## 4.1 Problem and History

Sorting integers is a recurring problem in computer science. The question is how fast we can sort n integers from $\{0, 1, \ldots, u-1\}$ sorting on word RAM with $w$-bit words, where $b = \lg u \leq w$.

Commonly used sorting algorithms and running times (linear space, unless otherwise specified):

- comparison sort - $O(n \lg n)$
- counting sort - $O(n+u)$, $O(n+u)$ space
- radix sort - $O(n \frac{\lg u}{\lg n})$
- van Emde Boas sort - $O(n \lg \lg u)$, $O(u)$ space

In 1995, a new sort was developed that achieved $O(n \lg \lg n)$ running time. This is a significant improvement because it achieves a faster running time than comparison sort, regardless of the size of the universe. Several improvements have been made, reducing space usage and improving the running time, but we will focus on the original algorithm.

$O(n \lg \lg n)$ sorting algorithms and running times (linear space, unless otherwise specified):

- Andersson, Hagerup, Nilsson, Raman (1995) - $O(n \lg \lg n)$, $O(\sqrt{u})$ space unless randomized
- Han (2002) - $O(n \lg \lg n)$ (deterministic)
- Han & Thorup (2003) - $O(n\sqrt{\lg \lg n})$ (randomized)
- Thorup (2003) - $O(nf(n))$ with $f(n) + O(1)$ priority queue (Insert, Delete, Find-Min)

## 4.2 Overview of $O(\lg \lg n)$ Randomized Sorting

The sorting algorithm described by Andersson et al. in 1995 acheived its running time through two main tools, range reduction and packed sorting.

**Range Reduction** The first tool, range reduction, reduces the problem of sorting $n$ numbers with $b$ bits to sorting $n$ numbers with $b/2$ bits in $O(n)$ time. Using randomization, it can be done in $O(n)$ space, $O(\sqrt{u})$ space is needed otherwise.

Applying range reduction $2 \lg \lg n$ times will reduce the number of bits from $b = \lg u$ to $\frac{b}{\lg^2 n}$. If our word-size $w \geq b \lg^2 n$, then we can fit $\lg^2 n$ elements in each word.

**Packed Sorting** Packed sorting, the second tool used in the sorting algorithm, allows for $O(n)$ sorting if the word-size $w \geq b \lg n \lg \lg n$ [1].

## 4.3 Range Reduction [Kirkpatrick & Reisch 1984]

Range reduction reduces sorting $n$ numbers of $b \leq w$ bits to sorting $n$ numbers of $b/2$ bits at a cost of $O(n)$ time.

There are four steps:

1. Cluster each number $x$ according to $high(x)$
   - For each $x$, add $x$ to list $B[high(x)]$
2. Form coordinate pairs and sort
   - Add coordinate pair $(high(x), *)$ for each distinct $high(x)$. This is easy if you maintain a list of nonempty $B$ lists.
   - Add coordinate pair $(low(x), high(x))$ except for $x = min(B[high(x)])$ [2] , computed by scanning through each $B$ list.
   - Sort pairs by first coordinate
3. Cluster sorted pairs by second coordinate (stably)
   - The $*$ cluster tells us the order of the $B$ lists
   - Each other cluster corresponds to a $B$ list and tells us the order of the list entries after the minimum
4. Concatenate $min(B[high])$ and $high$ cluster for each $high$ in order from $*$ cluster

Range reduction uses $O(\sqrt{u})$ space for $B$ and cluster arrays. It is possible to cluster together like values using hashing in $O(n)$ time and space.

[1] Different randomize range reduction, packed sort and "signature" sort have also been used to sort in $O(n)$ time when $w \geq (\lg n)^{2+\epsilon}$

[2] van Emde Boas trick to keep the number of entries being sorted down to $n$

## 4.4 Carrying Auxiliary Data

Range reduction reduces the sorting problem by sorting coordinate pairs by the firs coordinate. It is necessary to move the second coordinate, the auxiliary data, with the first as we sort. To do this without loss of efficiency, require sort to return a permutation of the original order and use the permuatation to look up auxiliary data.

For packed sorting, append element's auxiliary data to each element. This requires increasing the number of bits needed to store each entry from $b$ to $2b$ and adds one more range reduction step to the overall algorithm.

## 4.5 Packed Sorting

Using word size $w \geq 2(b+1)\lg n \lg\lg n$[3] we can sort in $O(n)$ time. This is possible because up to $2\lg n \lg\lg n$ elements can be packed into each word. Canonically, pack $\lg n \lg\lg n$ elements per word.

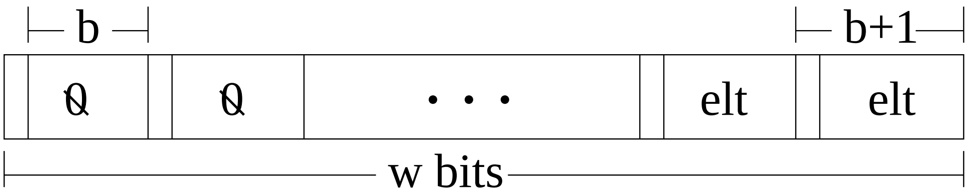


Figure 4.1: Elements in a packed word are $b$-bits long and preceded by a bit of padding. Half of the elements are left empty, indicated by the 0.

Steps for packed sorting:

1. Merge a pair of sorted words with $k \leq \lg n \lg\lg n$ entries into one osrted word with $2k$ entries in $O(\lg k)$ time. This is the hardest step of the packed sorting algorithm. It is accomplished through a combination of bitonic sort and bit tricks.

   $O(\lg k)$ time

2. Merge sort $k = \lg n \lg\lg n$ elements and pack into one word in $O(k)$ time. This is accomplished by recurively merging using Step 1. The recurrence can be written

$$T(k) = 2T(k/2) + O(\lg k).$$

   The base case is

$$T(2) = O(1).$$

[3]We use $b+1$ instead of b because it will prove convenient to have a bit of padding before each entry

This gives us the solution

$$T(k) = O(k).$$

$O(k)$ time

3. Merge two sorted lists of $r$ sorted words into one sorted list of $2r$ sorted words in $O(r \lg k)$ time. This is the standard merge algorithm using Step 1 as comparator.

$O(r \lg k)$ time

4. Merge sort $n$ elements with Step 3 as the merge procedure and Step 2 for the base case. The recurrance can be written

$$T(n) = 2T(n/2) + O(n/k \lg k)$$

from Step 3, where $r = n/k$ and

$$T(k) = O(k)$$

from step 2. Solving, we find

$$T(n) = O(n/k \lg k \lg n/k + kn/k)$$
$$T(n) \le O(n/k \lg k \lg n + n)$$

Substitute $k = \lg n \lg\lg n$

$$T(n) = O(\tfrac{n}{\lg n \lg\lg n}(\lg\lg n + \lg\lg\lg n) \lg n + n)$$
$$T(n) = O(n)$$

$O(n)$ time

## 4.6 Bitonic Sorting

A sequence of numbers is bitonic if it is a cyclic shift of nondecreasing and nonincreasing numbers:

1 2 3 4 5 6

6 5 4 3 2 1

4 5 6 1 2 3

2 1 6 5 4 3

The bitonic sorting algorithm sorts such a sequence into a nondecreasing sequence.

- Put $A[i]$ and $A[k/2+i]$ in the correct order for $i = 0, 1, \ldots, k/2$
- Split $A$ in half (at $k/2$)
- Recurse on halves (in parallel)
- Concatenate

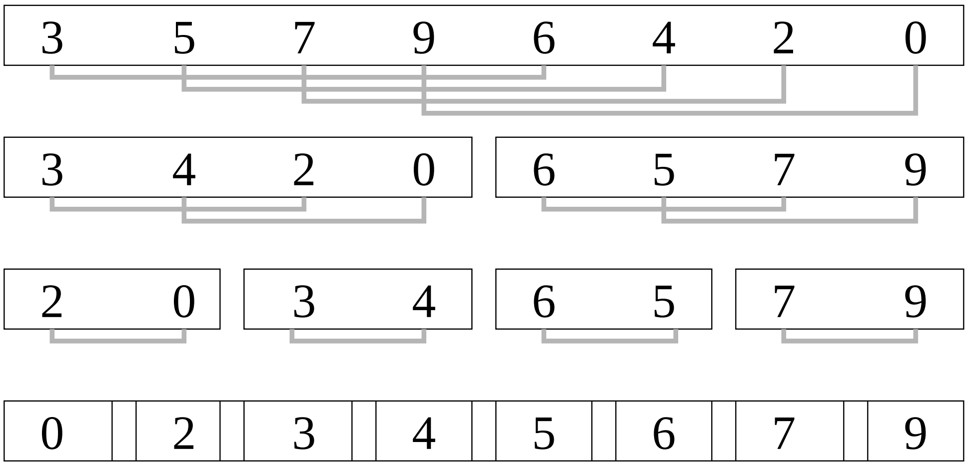


Figure 4.2: Bitonic sort on a sequence of eight numbers requires three rounds. The grey lines indicates which numbers are compared and considered for swapping.

After each round, the two halves are bitonic and the left half is smaller than the right[4]. It take $O(\lg k)$ rounds to sort the entire sequence.

## 4.7 Bit Tricks

To achieve $O(\lg k)$ merging of two sorted words, each with $k$ entries, we will need to find a way to do each of the $O(\lg k)$ rounds of bitonic sort in $O(1)$ time[5]. This is accomplished through a series of word operations:

[4]See CLRS for a proof of this

[5]We will also need to create a bitonic sequence from two nondescending sequences in $O(\lg k)$ time, but that is left as an exercise to the reader.

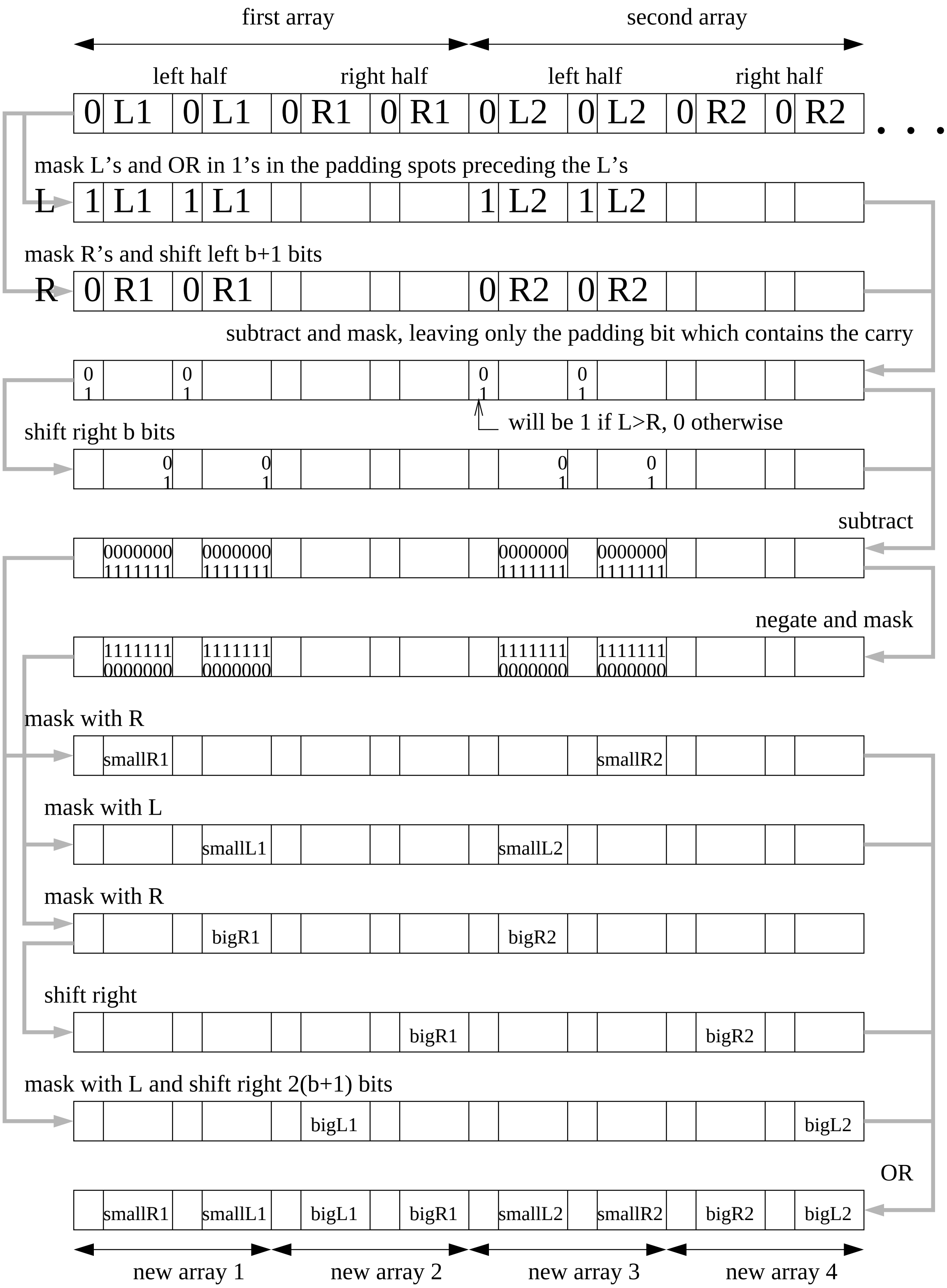


Figure 4.3: One round of bitonic sort can be completed in a constant amount of time using word operations. Blank spaces in the arrays are full of zeros.