

4. Non Adaptive Sorting Batcher's Algorithm

4.1 Introduction to Batcher's Algorithm

Sorting has many important applications in daily life and in particular, computer science. Within computer science several sorting algorithms exist such as the "bubble sort," "shell sort," "comb sort," "heap sort," "bucket sort," "merge sort," etc. We have actually encountered these before in section 2 of the course notes.

Many different sorting algorithms exist and are implemented differently in each programming language. These sorting algorithms are relevant because they are used in every single computer program you run ranging from the unimportant computer solitaire you play to the online checking account you own. Indeed without sorting algorithms, many of the services we enjoy today would simply not be available.

Batcher's algorithm is a way to sort individual disorganized elements called "keys" into some desired order using a set number of comparisons. The keys that will be sorted for our purposes will be numbers and the order that we will want them to be in will be from greatest to least or vice a versa (whichever way you want it).

The Batcher algorithm is non-adaptive in that it takes a fixed set of comparisons in order to sort the unsorted keys. In other words, there is no change made in the process during the sorting. Unlike other methods of sorting where you need to remember comparisons (tournament sort), Batcher's algorithm is useful because it requires less thinking. Non-adaptive sorting is easy because outcomes of comparisons made at one point of the process do not affect which comparisons will be made in the future. The algorithm simply runs and then eventually everything is sorted.

In order to visualize the following explanation of Batcher's algorithm imagine that a line of boxes exist with a single key in each of the boxes. The boxes describe place holders or hold some sort of rank (the first box is supposed to house the largest key etc.). Given two keys "x" and "y" located in boxes "a" and "b" respectively, we will compare the keys by taking both of them out of the two boxes and putting the larger key into box "a" (smaller one in box "b").

The algorithm can be described as the following notation...

New a = max(old a, old b)

New b = min(old a, old b)

In Excel this operation is "=max(cell1,cell2)" and "=min(cell1,cell2)" with the original value of the first key in cell1 and the original value of the second key in cell2.

The Batcher Algorithm uses a series of the above operation in order to sort a pile of unsorted keys. Batcher's algorithm works under the assumption that you have a "pile" of keys or simply a bunch of keys that you start off with that may or may not be sorted. If you have a list of keys arranged from left to right, and you sort the left and right halves of

the list separately, and then sort the keys in the even positions of the list, and in the odd positions separately, then all you need to do is compare and switch each even key on the left with the odd key immediately to its right and you will have completely sorted the whole list. We will see why this works in section 4.2. Note each sort procedure (evens and odds) uses the Batcher's algorithm as well.

Look at the following example of sorting 8 keys. The following charts are merely for a conceptual understanding of what is going on. The multiple operations that would take to execute the actual algorithm will be part of the problem set that will be given out. By understanding the following arguments we can figure out the Batcher algorithm. We first start off with this list of keys in the following positions...

Position	1	2	3	4	5	6	7	8
Key	1	7	3	4	5	2	8	6

Sorting the first and second halves separately changes this to the following chart. The bold signifies that one set of elements was sorted in increasing order while the rest of the un-bolded numbers are the other set of elements sorted in increasing order. In the chart below the keys in position 1-4 were sorted in increasing order and likewise the keys in positions 5-8 underwent the same operation. The way that these keys were sorted was that positions 1, 2 and 3, 4 were compared with each other putting the larger key in the higher position. We then accomplish the next step of merging where we compare and switch the new keys in positions 1, 3, and 2, 4. After this merge step made the new keys in position 2 and 3 were compared again placing the larger key in the higher position. We end up with 1, 3, 4, and 7 in the bold set of keys/positions. The same thing was done in the non-bold set.

Position	1	2	3	4	5	6	7	8
Key	1	3	4	7	2	5	6	8

Sorting the odd and even placed positions separately changes this to the following chart. Again the bold signifies the same thing as before meaning the bolded keys are sorted amongst each other as a set. They keys in position 1, 3, 5, and 7 are rearranged in increasing order and the same thing is done with positions 2, 4, 6, and 8. This was done by comparing and switching keys in positions 1, 3, and 5, 7, switching keys in positions 1, 5, and 3, 7 and finally switching the new keys in position 3, 5. This takes care of the odd half of the sort. For the even half of the sort positions 2, 4 and 6, 8 were compared and switched, then pairs 2, 6 and 4, 8, and finally pairs 4, 6 were compared and switched.

Position	1	2	3	4	5	6	7	8
Key	1	3	2	5	4	7	6	8

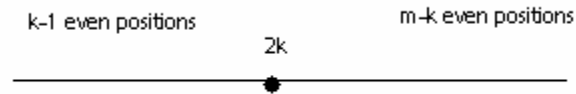
Comparing and switching the 2nd to 7th positions will then finish the sorting algorithm. This is because the first position and the last position should respectively have the lowest and highest numbered key. In sum we have completed "left and right half" position sorts and "odd and even" position sorts. The "left and right half" position sort

refers to comparing and sorting the keys in the first half of the positions and in the second half of the positions. The “odd and even position sorts” refer to sorting the keys in odd positions and in even positions.

4.2 Explanation of why the Batcher Algorithm Works

Consider a key in an even position $2k$ after completing the first and odd even position sort, and suppose that whenever we compare and switch it is the smaller key that goes to the left.

If we are trying to sort $2m$ keys, then there are $k - 1$ even position keys to its left and $m - k$ even position keys to its right.



Each even position key had a unique odd position key to its immediate left that had to be smaller than it after the first round of left right sorts: there are therefore at least k odd position keys that are smaller than the key in position $2k$ which makes $2k-1$ smaller keys all together. After all the sorting is complete this key belongs at position $2k$ or to its right.

Similarly each odd position key has an even position key that is larger immediately to its right after the left and right half sorts. After a complete sort it will be in its position or somewhere to its left.

If we were to do a similar count of keys that belonged to the right of the $2k^{\text{th}}$ key we find that each even position key except the top key on the left and the top key on the right has an odd key to its right.

If the keys were completely sorted we would know the exact sorted position of the key in the $2k^{\text{th}}$ position and the same count would differ by exactly one; each even position key has an odd larger key except for the last key.

We can conclude that the count of keys to the right of the $2k^{\text{th}}$ key in our case is at least $2m - 2k - 2 + 1$ instead of $2m-2k$ we would have if the list were completely sorted.

The extra ones comes from the fact that in our case there is a top key on the left and to the right, while in the sorted case there is only one rather than two top keys. This means that our $2k^{\text{th}}$ position key after the left right and odd even sorts is either where it belongs or one place to the left of where it belongs.

An odd position key is similarly either where it belongs or one place to the right of it. The comparing and switching each even key with the odd key to the right will finish the sorting job.

4.3 Figuring out the Batcher Algorithm

The previous facts tell us how to sort $2m$ keys if we can sort m keys. We can sort the left and right sets of the m keys, we can conduct the odd and even sort, and then do the last round of even and odd comparisons and switches.

The algorithm is actually simpler because after we sort the left and right halves of the keys, the left and right halves of both the even and odd keys are already sorted, so we don't have to do the sort lefts and rights in sorting the even and odd keys.

The following is the general outline of the Batcher Algorithm consisting of the sorting and merging of lists.

Sort Algorithm for $2m$ keys = Sort the left and half sides of the list. Then merge the two halves.

Merge $2m$ keys = Merge m odd keys and m even keys. Then compare and switch each even key with odd key to its right.

Comparing and switching two keys can be described by an ordered pair of key positions which can be described by a directed edge of a directed graph. The edge will point from the position to get the smaller key into the one that gets the larger one.

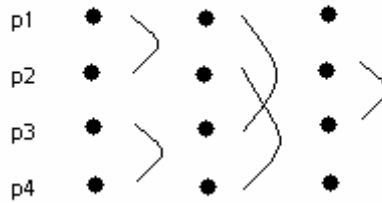
Suppose for example that we want to sort our initially unsorted keys into pairs. The algorithm for this is not what we have been describing but it is easier. You compare and switch them. This is exactly the same as using the "max" and "min" commands in excel spreadsheets.

4.4 Methodology for the Batcher Algorithm

Suppose now that we want to sort four keys in positions "p1," "p2," "p3" and "p4." This is the sort 4 algorithm.

We first sort the left and right halves or "pairs" describe as the edges (p1, p2) and (p3, p4). The notation for (p1, p2) means that we compare and switch the keys in position p1 and p2. The way you want to sort the keys from increasing or decreasing position is completely up to you as is the way you define which positions are hierarchically ordered. Then we merge by sorting the odd and evens with edges (p1, p3) and (p2, p4) and then completing (p2, p3). In the diagram below, each of the curved lines is a comparison and each dot is a key. The subsequent line of dots is the position of the keys after the comparisons are made. The first column of dots is the 1st stage of comparison and the second is the second stage etc.

Sort 4 Algorithm



We will sort 8 keys this way, and leave the 16 and 32 cases to you for the problem set.

The sort 8 keys we first apply the sort 4 algorithm the first half and the second half of our keys, separately. This takes the same three rounds of comparison switches just described. We described this exact procedure in section 4.1 too.

Sort Pairs:

(p1, p2) (p3, p4) (p5, p6) (p7, p8)

Merging Step1:

(p1, p3) (p2, p4) (p5, p7) (p6, p8)

Merging Step2:

(p2, p3) (p6, p7)

Now we repeat the merge 4 steps on the odd and even positions separately

Step 1: (p1, p5) (p2, p6) (p3, p7) (p4, p8)

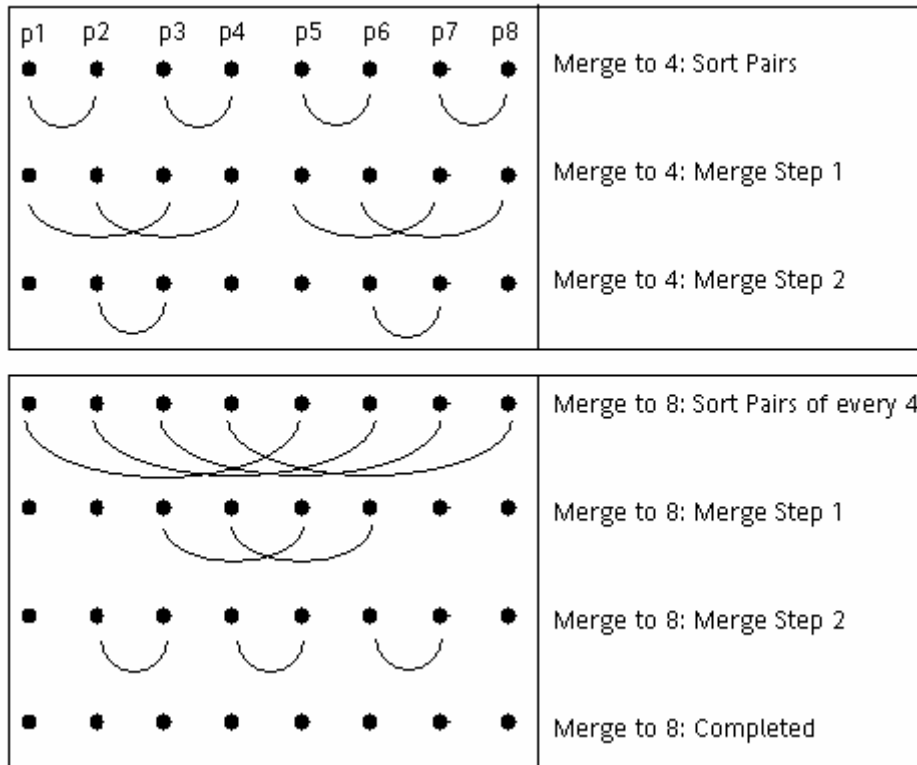
Step 2: (p3, p5) (p4, p6)

And finally compare and switch evens with the next odd positions

(p2, p3) (p4, p5) (p6, p7)

Notice that repeating steps on evens and odds, has the effect of doubling the distance between the front and back of each edge and making each edge into two parallel overlapping edges. Here is the diagram for the sort 8 algorithm.

Sort 8 Algorithm



One nice feature of this algorithm which is perhaps not so nice for you, is that it is easy to implement this algorithm on a spreadsheet, where you can put the original keys in the first column, put the comparison switches in the subsequent columns (using say $b1 = \max(a1, a2)$ and $b2 = a1 - b1 + a2$ for a comparison and switch). You can then watch your keys get sorted.

Exercises

- Exercise 1* How many “rounds” of comparison switches (not using the same key twice in any one round) are required to sort 2^k keys? How many comparison switches are used in this algorithm? You need not get the answer exactly, but will get extra credit for an exact answer.
- Exercise 2* Draw a graph of each of the rounds needed to sort 16 keys here. This is simply diagramming out the switches etc.
- Exercise 3* Setup a spreadsheet that implements Batcher’s algorithm for 32 keys. You can do this through entering a small number of commands in excel and copying them. (hint: some of the steps repeat themselves)
- Exercise 4* Exactly how many comparison switches are needed for 16 keys? Compare that number with the number needed to insert or simple merge sort.