

- Describe an algorithm to compute the maximum overlap length between any pair of *distinct* intervals I_i and I_j . You may assume the intervals are given by two arrays $A[1..n]$ and $B[1..n]$ where $A[i] = a_i$ and $B[i] = b_i$ for all $i \in \{1, \dots, n\}$. You may assume all array values are distinct.

Solution: The procedure $\text{MAXOVERLAP}(A[1..n], B[1..n])$ given below computes the maximum overlap length between pairs of distinct intervals given by A and B **assuming A is sorted in ascending order.**

```

MAXOVERLAP( $A[1..n], B[1..n]$ ):
  ⟨⟨Make sure we don't use the solution to meaningless base cases.⟩⟩
  if  $n < 2$ 
    return  $-\infty$ 

   $m \leftarrow \lfloor n/2 \rfloor + 1$ 
   $\text{leftOnly} \leftarrow \text{MAXOVERLAP}(A[1..m-1], B[1..m-1])$ 
   $\text{rightOnly} \leftarrow \text{MAXOVERLAP}(A[m..n], B[m..n])$ 

   $\text{maxReach} \leftarrow 1$ 
  for  $i \leftarrow 2$  to  $m-1$ 
    if  $B[i] > B[\text{maxReach}]$ 
       $\text{maxReach} \leftarrow i$ 
   $\text{across} \leftarrow 0$ 
  for  $j \leftarrow m$  to  $n$ 
     $\text{withJ} \leftarrow \max\{0, \min\{B[\text{maxReach}], B[j]\} - A[j]\}$ 
    if  $\text{withJ} > \text{across}$ 
       $\text{across} \leftarrow \text{withJ}$ 

  return  $\max\{\text{leftOnly}, \text{rightOnly}, \text{across}\}$ 

```

For any interval I_j , the best choice of interval I_i with $a_i < a_j$ is one where b_i is maximized.

Our algorithm for the stated problem begins by sorting A in ascending order using mergesort and permuting B to match. Then, we call $\text{MAXOVERLAP}(A[1..n], B[1..n])$ and return the result.

The running time for the call to MAXOVERLAP follows the mergesort recurrence $T(n) = 2T(n/2) + O(n)$, the solution to which is $O(n \log n)$. Together with the time to sort, **the algorithm takes $O(n \log n)$ time overall.** ■

Rubric: 10 points: standard recursive algorithm rubric.

2. Describe an algorithm to compute the weighted median of a given weighted set. You may assume your input consists of two *unsorted* arrays $S[1..n]$ and $W[1..n]$ where for each index i , the i th element of the set has value $S[i]$ and weight $W[i]$.

Solution: The procedure $\text{WEIGHTEDSELECT}(S[1..n], W[1..n], t)$ given below computes the element x where $w(S_{<_x}) \leq t$ and $w(S_{>_x}) < w(S) - t$ assuming $0 \leq t \leq w(S)$ (where $w(S)$ is the total weight of all elements 1 through n as described in the problem statement).

It relies on subroutine $\text{MOMSELECTPARTITIONW}(S[1..n], W[1..n], k)$ which permutes S and W identically and such that for all $i < k$, $S[i] < S[k]$ and for all $i > k$, $S[i] > S[k]$. It can be implemented by changing the procedure $\text{MOMSELECT}(S[1..n], k)$ seen in class and Erickson's algorithms to perform identical swaps to W as are applied to S and to do *nothing else*, because $\text{MOMSELECT}(S[1..n], k)$ already permutes the elements of S in the desired way. $\text{MOMSELECTPARTITIONW}$ still runs in $O(n)$ time.

```

WEIGHTEDSELECT( $S[1..n], W[1..n], t$ ):
  if  $n = 1$ 
    return  $S[1]$ 

   $k \leftarrow \lceil n/2 \rceil$ 
   $med \leftarrow \text{MOMSELECTPARTITIONW}(S[1..n], W[1..n], k)$ 

   $preWeight \leftarrow 0$ 
  for  $i \leftarrow 1$  to  $k - 1$ 
     $preWeight \leftarrow preWeight + W[i]$ 

  if  $t < preWeight$ 
    return  $\text{WEIGHTEDSELECT}(S[1..k-1], W[1..k-1], t)$ 
  else if  $t \geq preWeight + W[k]$ 
    return  $\text{WEIGHTEDSELECT}(S[k+1..n], W[k+1..n], t - preWeight - W[k])$ 
  else
    return  $S[k]$ 

```

Our algorithm for the stated problem calls $\text{WEIGHTEDSELECT}(S[1..n], W[1..n], w(S)/2)$.

The running time for the call to WEIGHTEDSELECT follows the recurrence $T(n) = T(n/2) + O(n)$, **the solution to which is $O(n)$** . ■

Rubric: 10 points: standard recursive algorithm rubric.

*3. Practice only. Do not submit solutions.

The following variant of the infamous StoogeSort algorithm¹ was discovered by the British actor Patrick Troughton during rehearsals for the 20th anniversary *Doctor Who* special “The Five Doctors”.²

```

WHOSORT( $A[1..n]$ ):
  if  $n < 13$ 
    sort  $A$  by brute force
  else
     $k = \lceil n/5 \rceil$ 
    WHOSORT( $A[1..3k]$ )      ⟨Hartnell⟩
    WHOSORT( $A[2k+1..n]$ )    ⟨Troughton⟩
    WHOSORT( $A[1..3k]$ )      ⟨Pertwee⟩
    WHOSORT( $A[k+1..4k]$ )    ⟨Davison⟩

```

- (a) Prove by induction that WHOSORT correctly sorts its input. [Hint: Where can the smallest k elements be?]

Solution: Let n be an arbitrary non-negative integer, let $A[1..n]$ be an arbitrary array, and assume that WHOSORT correctly sorts any array of size less than n . Assume that $n \geq 13$, since otherwise correctness is trivial.

Intuitively, WHOSORT partitions the input array A into five chunks, four of size $k = \lceil n/5 \rceil$ and one of size $n - 4k \leq n/5 \leq k$:

$A[1..k]$, $A[k+1..2k]$, $A[2k+1..3k]$, $A[3k+1..4k]$, $A[4k+1..n]$

The inequality $n \geq 13$ implies $4\lceil n/5 \rceil \leq n$, so all five chunks lie within the bounds of the original input array. (When $n = 16$, the last chunk is empty, but that’s not a problem.) Each recursive call to WHOSORT processes three consecutive chunks, which comprise a subarray of size at most $3k$. The assumption $n \geq 13$ also implies $3\lceil k/5 \rceil < n$, so by the induction hypothesis, the recursive calls correctly sort their respective subarrays.

Call an element of the input array **small** if it is one of the k smallest elements, **large** if it is one of the $n - 4k \leq k$ largest elements, and **medium** otherwise. Consider the locations of these classes of elements after each recursive call to WHOSORT.

- The inductive hypothesis implies that Hartnell moves all small elements in the first three chunks to chunk 1, and moves all large elements in the first three chunks to chunk 3. Thus, after Hartnell’s sort, all small elements are in chunks 1, 4, and 5, and all large elements are in chunks 3, 4, and 5.
- The inductive hypothesis implies that after Troughton’s sort, all small elements are in chunks 1 and 3, and all large elements (and nothing else)

¹https://en.wikipedia.org/wiki/Stooge_sort

²Tom Baker, the fourth Doctor, declined to return for the reunion; hence, only four Doctors appeared in “The Five Doctors”. (Well, okay, technically the BBC used excerpts of the unfinished episode “Shada” to include Baker, but he wasn’t really *there*—to the extent that any fictional character in a television show about a time traveling wizard arguing with several other versions of himself about immortality can be said to be “really” “there”.)

are in chunk 5. Moreover, chunk 5 is sorted; all large elements are in their correct final positions. The rest of the algorithm does not modify chunk 5.

- The inductive hypothesis implies that after Pertwee's sort, all small elements (and nothing else) are in chunk 1, in sorted order; thus, all small elements are in their correct final positions. The rest of the algorithm does not modify chunk 1. At this point, chunks 2, 3, and 4 contain all the medium elements and nothing else.
- Finally, the inductive hypothesis implies that Davison sorts chunks 2, 3, and 4, moving all the medium elements into their final correct positions.

We conclude that WHO SORT correctly sorts the array $A[1..n]$, as claimed. ■

I hate good wizards in fairy tales; they always turn out to be him.

— River Song [Alex Kingston], “The Pandorica Opens”, *Doctor Who* 5(12): 2010.

- (b) Would WHO SORT still sort correctly if we replaced “if $n < 13$ ” with “if $n < 4$ ”? Justify your answer.

Solution: No. When $n = 6$, the modified algorithm would fall into an infinite loop. Specifically, $k = \lceil 6/5 \rceil = 2$, so the first recursive call would attempt to recursively sort the entire array. ■

Solution: No. When $n = 11$, we have $k = \lceil 11/5 \rceil = 3$, so the last recursive call would attempt to recursively sort the “subarray” $A[4..12]$, which exceeds the bounds of the original input array. ■

- (c) Would WHO SORT still sort correctly if we replaced “ $k = \lceil n/5 \rceil$ ” with “ $k = \lfloor n/5 \rfloor$ ”? Justify your answer.

Solution: No. The modified algorithm would fail when $n = 14$ (and therefore $k = 2$), because there is not enough overlap between the recursive subarrays to carry all large elements to the last chunk. For example, the algorithm would modify the array **NMLKJIHGFEDCBA** as follows:

Sort $A[1..6]$	<u>IJKLMN</u> HGFEDCBA
Sort $A[5..14]$	IJKL <u>ABCDEF</u> GHMN
Sort $A[1..6]$	<u>ABIJKL</u> CDEFGHMN
Sort $A[3..8]$	ABCD <u>IJKLE</u> FGHMN

In contrast, the original algorithm (with $k = 3$) sorts the same array as follows.

Sort $A[1..9]$	<u>FGHIJKLM</u> NEDCBA
Sort $A[7..14]$	FGHIJK <u>ABCDE</u> LMN
Sort $A[1..9]$	<u>ABCFGHI</u> JKDELMN
Sort $A[4..12]$	ABCDEF <u>GHIJKL</u> MN

■

- (d) What is the running time of WHO SORT? (Set up a running-time recurrence and then solve it, ignoring the floors and ceilings.)

Solution: The running time obeys the recurrence $T(n) = 4T(3n/5) + O(1)$. The level sums of the recursion tree form an increasing geometric series, so the solution is $T(n) = O(4^{\log_{5/3} n}) = O(n^{\log_{5/3} 4}) = O(n^{2.7138309})$. ■

- (e) Forty years later, 15th Doctor Ncuti Gatwa discovered the following optimization to WHO SORT, which uses the standard MERGE subroutine from mergesort, which merges two sorted arrays into one sorted array.

```

<<Sort A>>
NuWHO SORT(A[1 .. n]) :
  if n < 13
    sort A by brute force
  else
    k = ⌈n/5⌉
    NuWHO SORT(A[1 .. 3k])           <<Grant>>
    NuWHO SORT(A[2k + 1 .. n])       <<Whittaker>>
    MERGE(A[1 .. 2k], A[2k + 1 .. 4k]) <<Tennant>>

```

What is the running time of NuWHO SORT?

Solution: The running time of NuWHO SORT obeys the recurrence $T(n) = 2T(3n/5) + O(n)$. The level sums of the recursion tree still form an increasing geometric series, but growing only half as fast as the recursion tree for WHO SORT. The solution is $T(n) = O(2^{\log_{5/3} n}) = O(n^{\log_{5/3} 2}) = O(n^{1.3569155})$.
And yes, the algorithm is still correct. ■