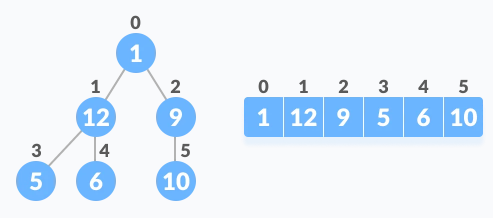


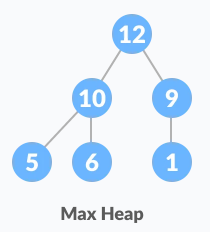
HEAP SORT ALGORITHM

The heap sort algorithm works by treating the array of numbers as a binary tree. There are 3 steps to the algorithm.

1. Make a max heap
2. Swap first and last elements of max heap
3. Reduce array size by 1

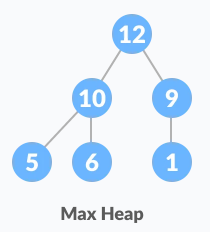


This is how a binary tree can be represented in an array. We can access the child nodes of a parent tree by the formula 2n+1 for the left child and 2n+2 for right. n is the parent node index.



This is an illustration of a max heap, where the parent nodes are greater than the child nodes.

This is after the sorting process has been done to sort the heap into a max heap.

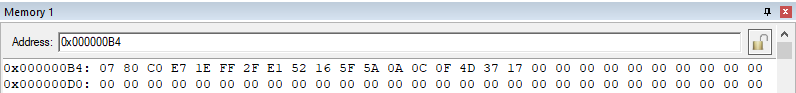
 This is after swapping the first and the last elements of the max heap.

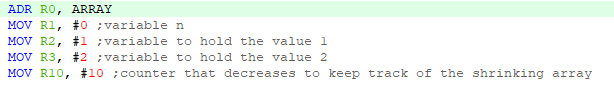
This entire process including the max heap sort is repeated until the entire list is sorted.

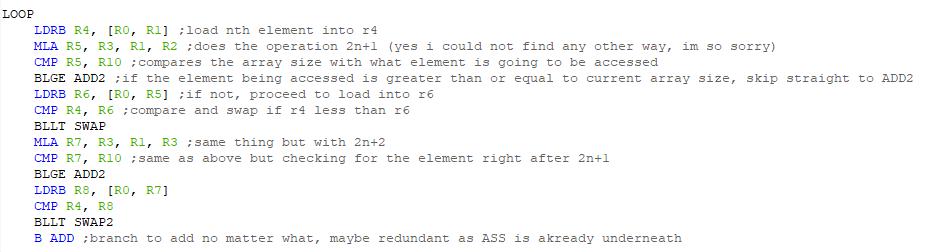
We start out by assigning the array of 10 numbers to the memory location:

0X000000BC to 0X000000C6





Then we access the address of the array and store it in R0 move constant values into registers R1, R2, R3 and R10.

This is the main body of the code that determines carries out the max heap sorting process

We load the first element from the array into R4 by using the offset value stored in R1, that value will be incremented each loop to iterate through the array.

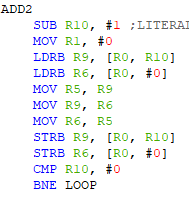
In the next line, the MLA operation carries out the formula 2n+1 by multiplying the value of R3 and R1, adding the result by R2 and then storing it in R5.



This is a safety check to ensure that the array element being accessed is still part of the array. In the event that the array has gone through at least one iteration of the sorting process, where there will be one less element in the array as a result, R10 represents the relative last element in the array if the offset being calculated, R5 is greater than R10, the current limit, then it branches straight to ADD2



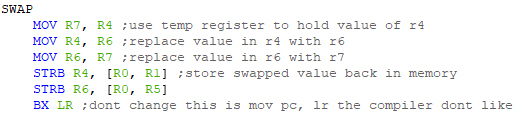
ADD2 decreases the value of R10 by 1 and resets the counter of R1. It also carries out the swapping process of the first and relative last element. It compares R10 to 0 and if it does not equal 0, it branches back to LOOP.



Assuming that the element being accessed by the offset was within limits, the code would have proceeded to:



Where it would load the left child element from the array with the offset of R5, compare it with the current parent element. If the parent element was less than the child, they would be swapped to create a max heap.

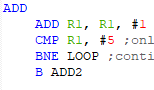






This is the swapping algorithm where it would swap the parent and left child elements being compared, store them back into memory and return the main body of code. It’s basically a subroutine.

The rest of the code progresses with the same logic as above, but it accounts for the formula 2n+2, meaning a different offset formula than the first one, 2n+1 and a different swap subroutine meant to swap parent and right child elements, SWAP2



This is the first ADD loop where it exists only to facilitate the program’s first iteration. After the first iteration where it loops 5 times, the counter never fully reaches 5 as it would cause the program to access elements that were outside the array. Which was why the second ADD2 loop was written to facilitate the safety check. When R10 finally decreases to 0, then the program stops, having sorted the entire array.