the end data ~~block~~ of ~~the~~ access requirement to ~~the~~ interior ~~and make it an interior block.~~ This will reduce its span, thus reducing the EST for the layout. ~~In order to maximize our benefit to our current access requirement, we want to copy the start data unit to somewhere between the one after the first one and the last one. In the same manner, we want to copy the end unit to somewhere between the first one and the one before the last one.~~

Figure 2 shows the access requirement from the above figure with an arrow showing the locations where the start unit can be moved to as well as what the access requirement looks like after the copy. By moving it to one of the these locations we are guaranteed to reduce the span of the access requirement. The dashed line connects the original data unit with its copy. The solid blue lines represent the new span of the blue access requirement. Because the start unit has been copied and its copy is being used, it no longer is needed and is thus not counted in the new span for the blue access requirement. The span of the blue access requirement has been reduced from 14 units to 12 units.

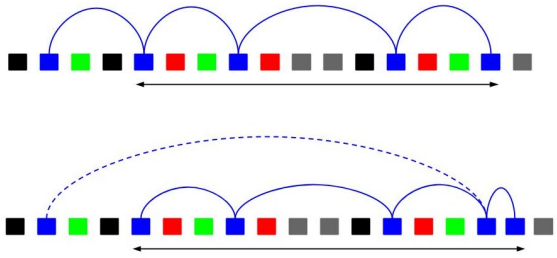

Figure 2: ~~*Interval where the start data unit can be copied to (top) and the data units after the copy (bottom)*~~

~~For the access requirement under consideration, it does not matter where in the specified interval our data unit is moved.~~ However, if the new location of the data ~~block~~ is in the span of other access requirements, it increases the seek time of those accesses by one unit. We thus want to find a location ~~that~~ in the ~~interior~~ of the access requirement under consideration but is in the span of least number of access requirements. Such a location is identified using a simple linear search through the span of the access requirement. Figure 3 shows the data units in figure 1 before the copying and highlights the gray access requirement that will be affected. As can be observed in figure 3 the gray access requirement has had its span increased by 1. Even though we reduced the total span by 2 with the blue access requirement, we also increased the total span by 1 with the gray access requirement, yielding a net benefit of 1 data unit. If ~~you~~ search through all the spots in the interior of the blue access requirement, then the spot chosen had the least number of overlapping access requirements at 1.

Moving versus Copying: A data block can be accessed by multiple access requirements. If that data block is an extremal block of access, and if it is moved to its interior, it may affect other access requirements that use the same data block. That is the main reason that we are copying and not moving these data blocks. By copying the data unit the other access requirements can still access it in its old location without any change in their span. Nevertheless, if by using the new copy the span of one or more of the other access requirements reduces, then those access requirements should use the new copy instead of the old copy. If all the access requirements use the new copy and the old data block is not used by any access, then it can be deleted. In this later case, we are really moving the data block.

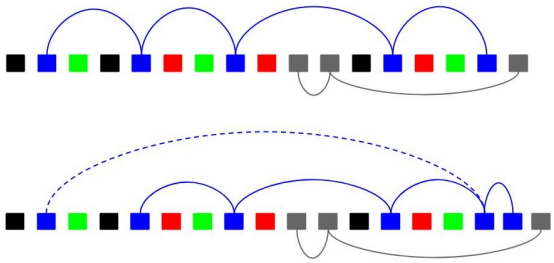

Figure 3: *The blue and gray access requirements before (top) and after (bottom) the copying*

Data Block processing order: We now need to figure out how to use this information to decide in what order the copying should be done. For each data unit, its total benefit is the amount that is reduces the total seek time (EST). For a given data unit to be copied to a specified location, let k_i be the benefit to access requirement i that is attached to the data unit. We will say that $k_i = 0$ if the access requirement will use the old copy and $k_i > 0$ if the access requirement will use the new copy. Let I be the set of access requirements attached to the data unit. Let J be the set of access requirements not in I whose span overlaps our data units. We can now describe the benefit as follows:

$$\textit{Benefit} = -\Delta EST = \sum_{i \in I} k_i - |J|$$

Before doing any actual copying, we compute the above described benefit for each start and end data unit for each access requirement. We store all the benefits into a binary search tree sorted in descending order by benefit amount. That way we will easily be able to choose the data unit that provides the most benefit in expected seek time. We will also make a special list L of cases where a data unit is copied then deleted because all the access requirements will benefit.

Because doing the moving for the list L does not increase the storage, we will first go through that list and perform those moves. Once that list is empty we will take the data unit in the binary search tree that provides the most benefit and perform the copy. We will then recompute L and the binary search tree. We will continue to do those steps until we have run out of available space for redundancy. As a summary, the psuedo-code of this algorithm is shown as algorithm 1.

4 Run-time and Storage Analysis

We now analyze the running time and storage requirements of our algorithm. We will denote N as the number of data units and A as the number of access requirements. We will use k as the average span of a single access requirement. The variable Q will represent how many executions of the redundancy loop occur. The average number of overlapping access requirements is proportional to the number of data units multiplied by the redundancy factor r, which is the amount of redundancy if we have a single-seek layout. Therefore, the number of overlapping access requirements is $O(rN)$.

4.1 Run-time Analysis

The loop that initially constructs the access requirement heap is run on all A access requirements. It involves finding all the benefit