1. Immediate coalescing effort includes when a free occurs, check the adjacent blocks to see if they are also free then combine them. Deferred coalescing effort includes only combining adjacent blocks at time of allocation. In terms of blocks immediate coalescing must free the current block and check both the block before and after to check if it can coalesce with either or both. Deferred coalescing will go through the free block list starting from the beginning and coalesce any adjacent free blocks after it that it has looked at both blocks, if only the first free block is looked at and used during allocation then nothing will be coalesced. Immediate coalescing would require more overall time because every time a block is freed it must check the block before and after, whereas deferred coalescing doesn’t combine any free blocks until it has looked at adjacent blocks upon allocation.
2. a) First, next, and best will all take the same amount of time in an empty heap.  
   b) Next worst case is if the last allocated block was at the beginning of the heap, otherwise next takes the shortest amount of time overall, picking up from the last allocated, making next the average the fastest time. First and best both take the same amount of time, best case is if there is enough space in front of the contiguous blocks, otherwise the average case is after the contiguous block.  
   c) Next has a best case assuming it starts in the middle of the heap. First has a best case of the first block being used for allocation. Next and first have the same average case that a freed block works before n number of freed blocks. First and next work have the fastest time when the needed size is small. Best has the best case that there is an exact fit early, otherwise it must check all free blocks, making best the worst time wise.  
   d) Next, first, and best are all bad here because with a full heap of small allocated blocks the freed blocks sizes are going to be small as well, large needed block sizes won’t work well. If the needed block size is small first and next have the average time as in c). Best if it doesn’t find an exact size, it must check all freed blocks.
3. Received signals not being queued matters because at times when we must handle all signals accordingly it can cause problems. Bugs may arise when a certain signal is pending and the kernel sends a more important signal, but since the process hasn’t received the first signal yet the second signal is discarded. We deal with this by using a signal handler. When a signal is received, the process pauses the program to handle the signal and when the handler has finished it returns and resumes the main program.
4. When setjmp is called and then longjmp is called anything queued into the stack between the two calls is “forgotten” and returns to the state of the original setjmp. Register context, stack pointer, and PC value are stored as part of the jmp\_buf.
5. Threads share data structures and the heap. Processes share a data object. Threads are more efficient than process and therefore are better for repeatedly interacting and communicating with each other. Threads can all share the same data and code and are more efficient than processes because there is no creating and reaping involved.
6. A=1, B=2, C=2, D=4, E=6
7. a) bac, abc  
   b) bca
8. 012
9. a), b), e)
10. d), e)