1. **First run with the flags -n 10 -H 0 -p BEST -s 0 to generate a few random allocations and frees. Can you predict what alloc()/free() will return? Can you guess the state of the free list after each request? What do you notice about the free list over time?**

./malloc.py -n 10 -H 0 -p BEST -s 0

```
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder ADDRSORT
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
compute False

ptr[0] = Alloc(3) returned 1000 (searched 1 items)
List[ Size 1]: [addr:1003 size:97]

Free(ptr[0])
returned 0
List[ Size 2]: [addr:1000 size:3] [addr:1003 size:97]

ptr[1] = Alloc(5) returned 1003 (searched 2 items)
List[ Size 2]: [addr:1000 size:3] [addr:1008 size:92]

Free(ptr[1])
returned 0
List[ Size 3]: [addr:1000 size:3] [addr:1003 size5][addr:1008 size:92]

ptr[2] = Alloc(8) returned 1008 (searched 3 items)
List[ Size 3]: [addr:1000 size:3] [addr:1003 size:5][addr:1016 size:84]

Free(ptr[2])
returned 0
List[ Size 4]: [addr:1000 size:3] [addr:1003 size5] [addr:1008 size:8][addr:1016 size:84]

ptr[3] = Alloc(8) returned 1008 (searched 4 items)
List[ Size 3]: [addr:1000 size:3] [addr:1003 size5][addr:1016 size:84]
```

Free(ptr[3])
returned 0
List[Size 4]: [addr:1000 size:3] [addr:1003 size5] [addr:1008 size:8][addr:1016 size:84]

ptr[4] = Alloc(2) returned 1000 (searched 4 items)
List[Size 4]: [addr:1002 size:1] [addr:1003 size5] [addr:1008 size:8][addr:1016 size:84]

ptr[5] = Alloc(7) returned 1008 (searched 4 items)
List[Size 4]: [addr:1002 size:1] [addr:1003 size5] [addr:1015 size:1][addr:1016 size:84]

2. **How are the results different when using a WORST fit policy to search the free list (-p WORST)? What changes?**

seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy WORST
listOrder ADDRSORT
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
compute False

ptr[0] = Alloc(3) 1000 (searched 1 items)
List[Size 1]: [addr:1003 size:97]

Free(ptr[0])
returned 0
List[Size 2]: [addr:1000 size:3] [addr:1003 size:97]

ptr[1] = Alloc(5) returned 1003 (searched 2 items)
List[Size 2]: [addr:1000 size:3] [addr:1008 size:92]

Free(ptr[1])
returned 0
List[Size 3]: [addr:1000 size:3] [addr:1003 size5][addr:1008 size:92]

ptr[2] = Alloc(8) returned 1008 (searched 3 items)
List[Size 3]: [addr:1000 size:3] [addr:1003 size:5][addr:1016 size:84]

Free(ptr[2])
returned 0
List[Size 4]: [addr:1000 size:3] [addr:1003 size5] [addr:1008 size:8][addr:1016 size:84]

ptr[3] = Alloc(8) returned 1016(searched 4 items)
List[Size 4]: [addr:1000 size:3] [addr:1003 size5] [addr:1008 size:8][addr:1024 size:76]

Free(ptr[3])
returned 0
List[Size 5]: [addr:1000 size:3] [addr:1003 size5] [addr:1008 size:8][addr:1016 size:8][addr:1024 size:76]

ptr[4] = Alloc(2) returned 1024
List[Size 5]: [addr:1000 size:3] [addr:1003 size5] [addr:1008 size:8][addr:1016 size:8][addr:1026 size:74]

ptr[5] = Alloc(7) returned 1026
List[Size 5]: [addr:1000 size:3] [addr:1003 size5] [addr:1008 size:8][addr:1016 size:8][addr:1033 size:67]

When alloc size 8 for ptr[3], instead use the fragment at addr:1008, it used the largest chunck at addr 1016 size:84. And this cause free ptr[3] have extra free space fragment of size 8 in the free list. Moreover, when alloc(2) for ptr[4] and alloc(7) for ptr[7], they always use the largest chunk instead the best 'fit' one.

3. **What about when using FIRST fit (-p FIRST)? What speeds up when you use first fit?**

seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy FIRST
listOrder ADDRSORT
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
compute False

ptr[0] = Alloc(3) returned 1000 (searched 1 items)
List[Size 1]: [addr:1003 size:97]

Free(ptr[0])
returned 0
List[Size 2]: [addr:1000 size:3] [addr:1003 size:97]

ptr[1] = Alloc(5) returned 1003 (searched 2 items)
List[Size 2]: [addr:1000 size:3] [addr:1008 size:92]

Free(ptr[1])
returned 0
List[Size 3]: [addr:1000 size:3] [addr:1003 size:5][addr:1008 size:92]

ptr[2] = Alloc(8) returned 1008 (searched 3 items)
List[Size 3]: [addr:1000 size:3] [addr:1003 size:5][addr:1016 size:84]

Free(ptr[2])
returned 0
List[Size 4]: [addr:1000 size:3] [addr:1003 size:5] [addr:1008 size:8][addr:1016 size:84]

ptr[3] = Alloc(8) returned 1008 (searched 3 items)
List[Size 3]: [addr:1000 size:3] [addr:1003 size5][addr:1016 size:84]

Free(ptr[3])
returned 0
List[Size 4]: [addr:1000 size:3] [addr:1003 size:5] [addr:1008 size:8][addr:1016 size:84]

ptr[4] = Alloc(2) returned 1000 (searched 1 items)
List[Size 4]: [addr:1002 size:1] [addr:1003 size:5] [addr:1008 size:8][addr:1016 size:84]

ptr[5] = Alloc(7) returned 1008 (searched 3 items)
List[Size 4]: [addr:1002 size:1] [addr:1003 size:5] [addr:1015 size:1][addr:1016 size:84]

We speed up the search time when we use FIRST fit, previously each time find new free memory we have to iterate entire free list to find the BEST/WORST fit memory chunk.

4. **For the above questions, how the list is kept ordered can affect the time it takes to find a free location for some of the policies. Use the different free list orderings (-l ADDRSORT, -l SIZESORT+, -l SIZESORT-) to see how the policies and the list orderings interact.**

The -l ADDRSORT is kept list in ascending order base on the address
The -l SIZESORT+ is kept list in ascending order base on the memory chunk size
The -l SIZESORT- is kept list in descending order base on the memory chunk size

5. Coalescing of a free list can be quite important. Increase the number of random allocations (say to -n 1000). What happens to larger allocation requests over time? Run with and without coalescing (i.e., without and with the -C flag). What differences in outcome do you see? How big is the free list over time in each case? Does the ordering of the list matter in this case?

Without -C coalescing flag the larger allocation over time the size of free list become larger. Moreover, a lot allocation return the -1 because small pieces of memory chunk didn't combine into large one which can satisfied large allocation of memory.
With -C coalescing flag the large allocation over time the size of free list can be kept in small size. Almost every large allocation of memory success, and the search time is fast since free list size is relatively small.
The reason is simple with the coalescing the free list have a chance to combine multiple neighbor free memory chunk into large one.

Yes, ordering is really important, because coalescing only available continuous chunk of memory. If we order them by the size of the free memory, then they might not in a continuous memory. Implies coalescing may never have chance to work

6.What happens when you change the percent allocated fraction -P to higher than 50? What happens to allocations as it nears 100? What about as the percent nears 0?

With -P higher than 50, the allocation() chance is higher than the free() chance, thus the total available memory will become smaller and smaller.

With -P 100 then every space is used up, and eventually no more space to allocate.
With -P 0(Raise assertion error), I'm guess that it will only doing free() and never allocate, thus will cause the errors.

7. What kind of specific requests can you make to generate a highly fragmented free space? Use the -A flag to create fragmented free lists, and see how different policies and options change the organization of the free list.

```
./malloc.py -p WORST -A +1,+1,+1,+1,+1,-0,-1,-2,-3,-4 -c
```

We use worst policy and never use coalescing to combine the free list. The reason we use worst is because we don’t want simulator reuse small freed memory fragment to allocate again, instead use largest chunk. Other police will just use smallest free memory fragment.