## Question 1

**First run with the flags:**
**Python3 malloc.py -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?**

```
ptr[0] = Alloc(3)
returned: 1000
List: [Size: 1]: [ a:1003 s:97 ]

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

ptr[1] = Alloc(5)
returned: 1003
List: [Size: 2]: [a:1000 s:3][a:1008 s:92]

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

ptr[2] = Alloc(8)
returned: 1008
List: [Size: 3]: [a:1000 s:3][a:1003 s:5][a:1016 s:84]

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

ptr[3] = Alloc(8)
returned: 1008
List: [Size: 3]: [a:1000 s:3][a:1003 s:5][a:1016 s:84]

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

ptr[4] = Alloc(2)
returned: 1000
List: [Size: 4]: [a:1002 s:1][a:1003 s:5][a:1008 s:8][a:1016 s:84]

ptr[5] = Alloc(7)
returned: 1008
List: [Size: 4]: [a:1002 s:1][a:1003 s:5][a:1015 s:1][a:1016 s:84]
```

```
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
```

Over time the free list will get pretty fragmented without coalescing.

## Question 2

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

**Python3 malloc.py -n 10 -H 0 -p WORST -s 0**

```
ptr[0] = Alloc(3)
returned: 1000
List: [Size: 1]: [a:1003 s:97]

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

ptr[1] = Alloc(5)
returned: 1003
List: [Size: 2]: [a:1000 s:3][a:1008 s:92]

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

ptr[2] = Alloc(8)
returned: 1008
List: [Size: 3]: [a:1000 s:3][a:1003 s:5][a:10016 s:84]

Free(ptr[2])
returned: 0
List: [Size: 4]: [a:1000 s:3][a:1003 s:5][a:1008 s:8][a:10016 s:84]

ptr[3] = Alloc(8)
returned: 1016
List: [Size: 4]: [a:1000 s:3][a:1003 s:5][a:1008 s:8][a:10024 s:76]

Free(ptr[3])
returned: 0
List: [Size: 5]: [a:1000 s:3][a:1003 s:5][a:1008 s:8][a:10016 s:8][a:1024 s:76]

ptr[4] = Alloc(2)
returned: 1024
List: [Size: 5]: [a:1000 s:3][a:1003 s:5][a:1008 s:8][a:10016 s:8][a:10026 s:74]

ptr[5] = Alloc(7)
returned: 1026
List: [Size: 5]: [a:1000 s:3][a:1003 s:5][a:1008 s:8][a:10016 s:8][a:1033 s:67]
```

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

The free list grows even faster, since we always use the worst fit.

## Question 3

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

```
ptr[0] = Alloc(3)
returned: 1000
List: [Size: 1]: [a:1003 s:97]

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

ptr[1] = Alloc(5)
returned: 1003
List: [Size: 2]: [a:1000 s:3][a:1008 s:92]

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

ptr[2] = Alloc(8)
returned: 1008
List: [Size: 3]: [a:1000 s:3][a:1003 s:5][a:1016 s:84]

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

ptr[3] = Alloc(8)
returned: 1008
List: [Size: 3]: [a:1000 s:3][a:1003 s:5][a:1016 s:84]

Free(ptr[3])
returned:
List: [Size: 4]: [a:1000 s:3][a:1003 s:5][a:1008 s:8][a:1016 s:84]

ptr[4] = Alloc(2)
returned: 1000
List: [Size: 4]: [a:1002 s:1][a:1003 s:5][a:1008 s:8][a:1016 s:84]

ptr[5] = Alloc(7)
returned: 1008
List: [Size: 4]: [a:1002 s:1][a:1003 s:5][a:1015 s:1][a:1016 s:84]
```

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

With the first fit strategy the OS doesn't need to iterate over the entire free list and just takes the first block that's big enough, thus making it faster then best fit and worst fit, but may result in more internal fragmentation.

## Question 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.**

ADDRSORT :
With the addresses sorted **none** of the strategies will have benefits, since this is only really helpful with coalescing

SIZESORT+ :
If the free blocks are sorted by size from small to big, the **first fit** and **best fit** strategy will profit a lot.

SIZESORT- :
With the free blocks sorted from big to small, the **worst fit** strategy will profit a lot.

## Question 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?**

At when the fragmentation keeps happening, bigger allocations will get rejected because we don't have a single block that's big enough.

Best fit:
The sorting of the list doesn't matter, since we always iterate over the entire list anyways. Because we reuse small spaces the big spaces stay in one piece and we can use the mem longer without running out of space.

Worst fit:
The sorting of the list doesn't matter, since we always iterate over the entire list anyways. Because we always use the biggest space and split it, the space is used up really fast and only small blocks remain which makes it impossible to allocate bigger blocks.

First fit:
With first fit the sorting is determining if it works more like best fit or more like worst fit.

## Question 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?**

At 50%, **half** the requests are free() **and half** are malloc().
Below 50% we still have **50% malloc()** requests since we can't free something that's not allocated.
Above 50% the memory will eventually **fill up** since we malloc() more than we free().

## Question 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 organisation of the free list.**

```
python3 ./malloc.py -n 6 -A
+1,-0,+2,-1,+3,-2,+4,-3,+5,-4,+6,-5,+7,-6,+8,-7,+9,-8,+10,-9,+11,-10,+12,-11,+13,-12 -c
```