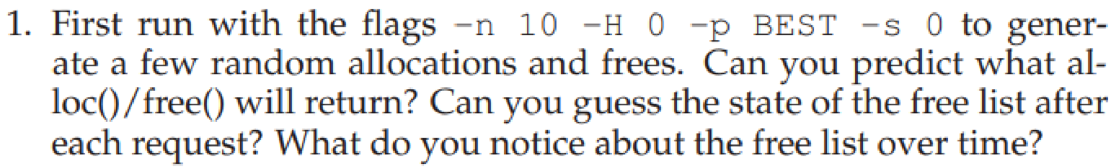
【OS】Day20(4)

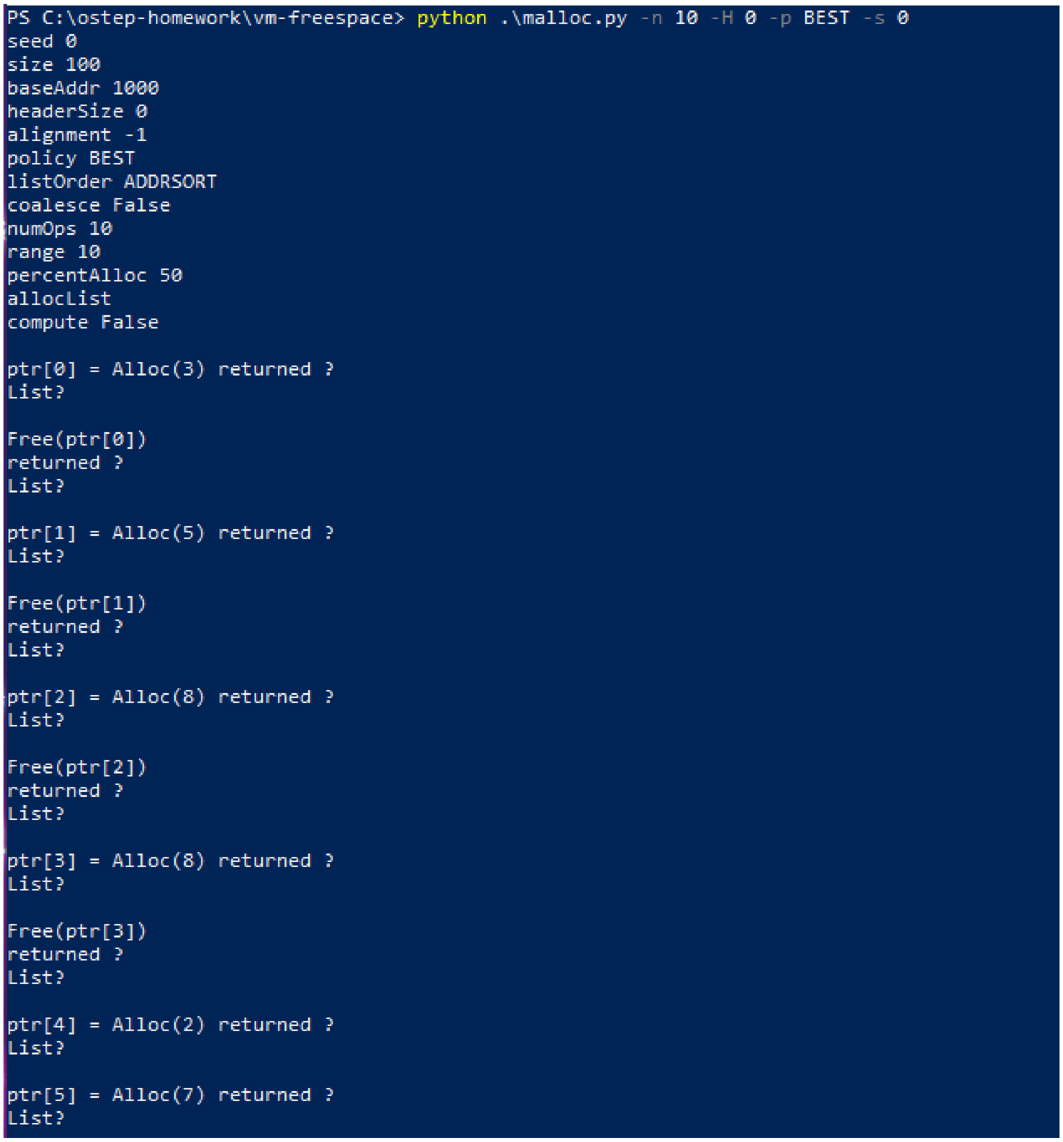
【Ch17】Free-Space Management

Question 1



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() will return the address of the allocated memory
- free() will not return anything



```
PS C:\ostep-homework\vm-freespace> python .\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 ?
List?

Free(ptr[0])
returned ?
List?

ptr[1] = Alloc(5) returned ?
List?

Free(ptr[1])
returned ?
List?

ptr[2] = Alloc(8) returned ?
List?

Free(ptr[2])
returned ?
List?

ptr[3] = Alloc(8) returned ?
List?

Free(ptr[3])
returned ?
List?

ptr[4] = Alloc(2) returned ?
List?

ptr[5] = Alloc(7) returned ?
List?
```

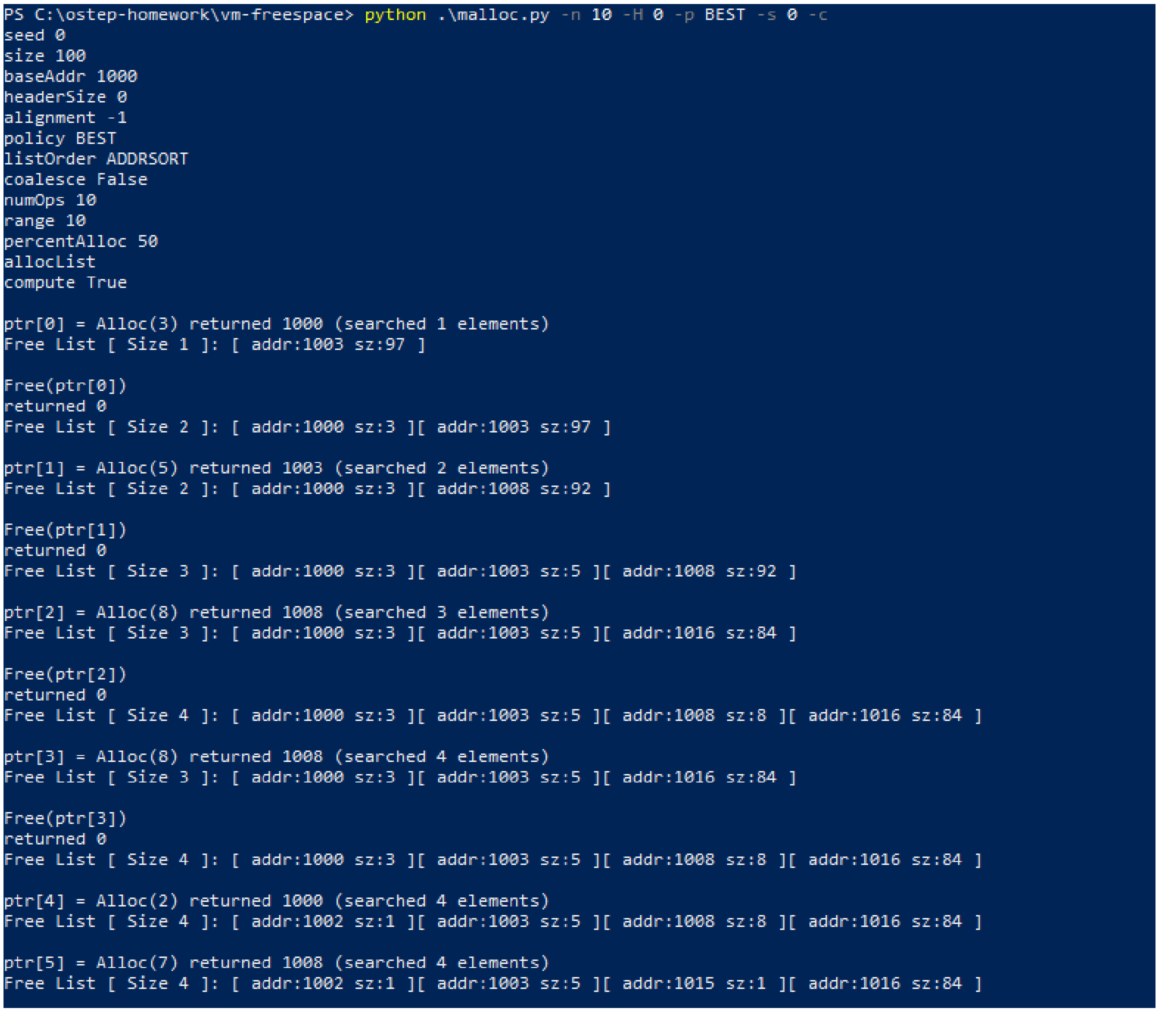
1. return: 1000 List: 1003 size: 97

2. return: void List: 1000 size 3 → 1003 size 97

3. return: 1003 List: 1000 size3 →1008 size92

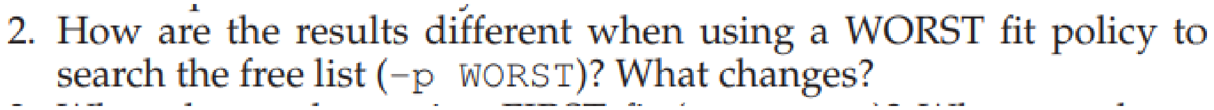
4. return: void List: 1000 size3→1003 size5 →1008 size92

5. The two malloc() instructions that ask for 8 bytes will reuse the newly-freed space.

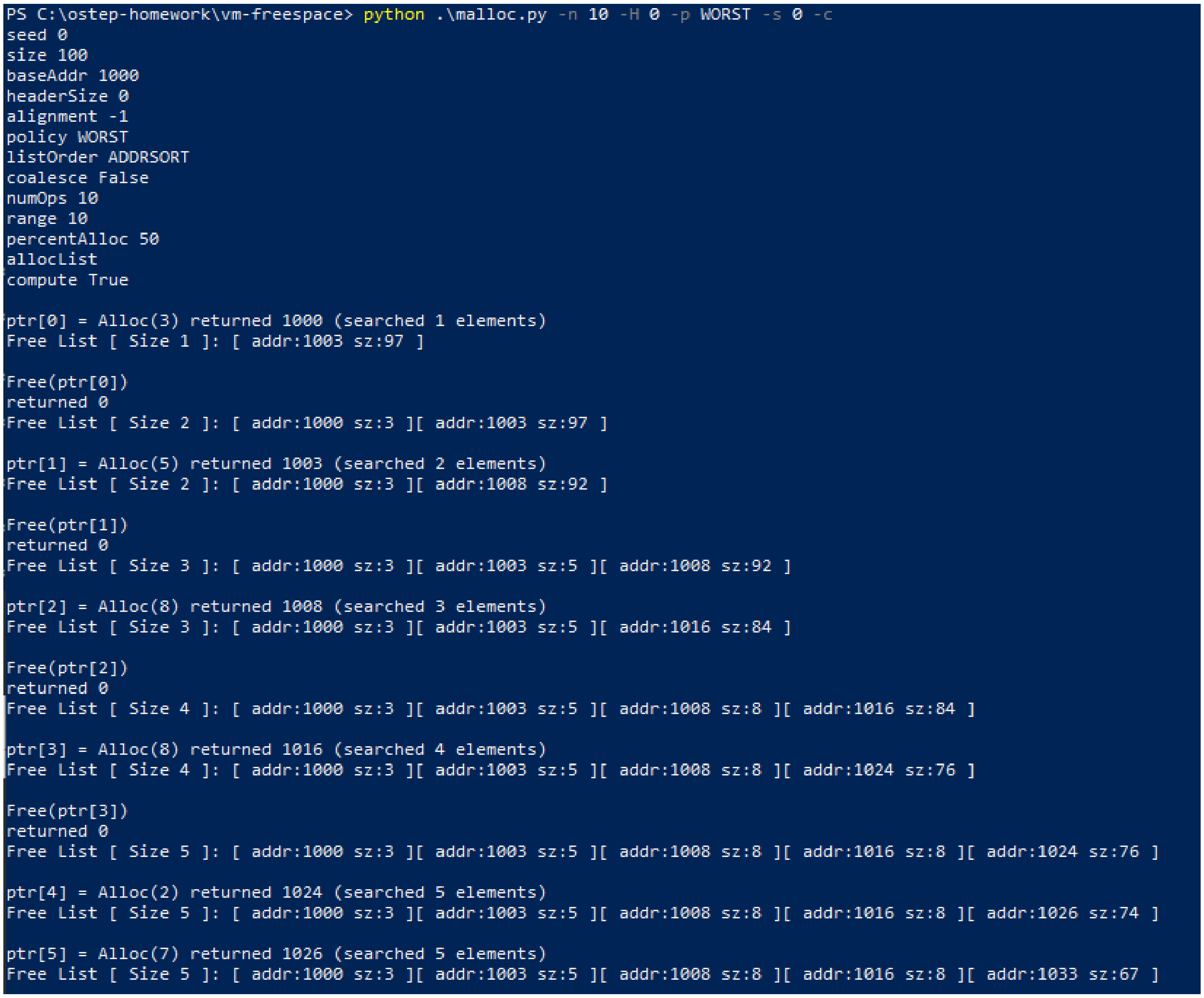


```
PS C:\ostep-homework\vm-freespace> python .\malloc.py -n 10 -H 0 -p BEST -s 0 -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder ADDRSORT
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
compute True

ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1003 sz:97 ]

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

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

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

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

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

ptr[3] = Alloc(8) returned 1008 (searched 4 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

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

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

ptr[5] = Alloc(7) returned 1008 (searched 4 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1015 sz:1 ][ addr:1016 sz:84 ]
```

Question 2



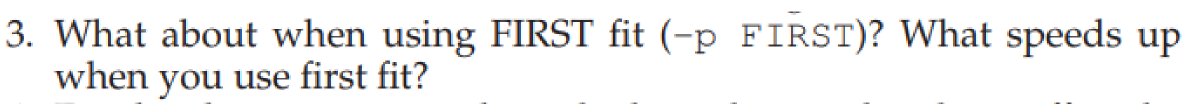
2. How are the results different when using a WORST fit policy to search the free list (-p WORST)? What changes?



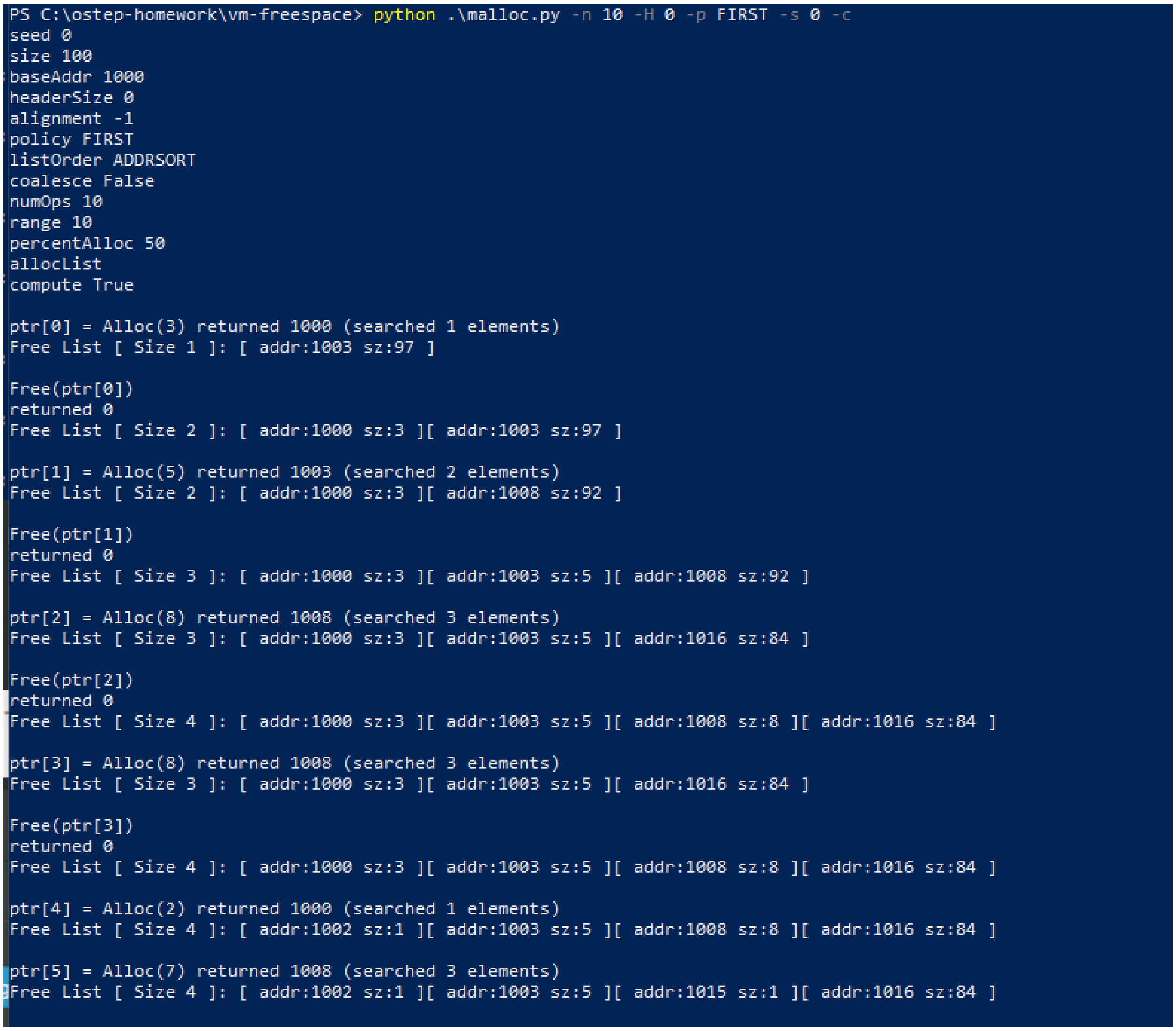
```
PS C:\ostep-homework\vm-freespace> python .\malloc.py -n 10 -H 0 -p WORST -s 0 -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy WORST
listOrder ADDRSORT
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
compute True

ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1003 sz:97 ]

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

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

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

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

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

ptr[3] = Alloc(8) returned 1016 (searched 4 elements)
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1024 sz:76 ]

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

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

ptr[5] = Alloc(7) returned 1026 (searched 5 elements)
Free List [ Size 5 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:8 ][ addr:1033 sz:67 ]
```

The allocator now will not reuse the 8-byte chunk freed by the previous instruction, which will create extra fragmentations.

Question 3

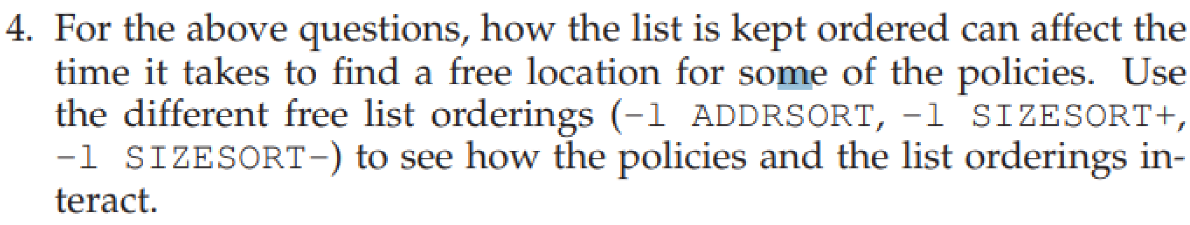


3. What about when using FIRST fit (`-p FIRST`)? What speeds up when you use first fit?



```
PS C:\ostep-homework\vm-freespace> python .\malloc.py -n 10 -H 0 -p FIRST -s 0 -c
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

ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1003 sz:97 ]

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

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

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

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

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

ptr[3] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

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

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

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

Question 4

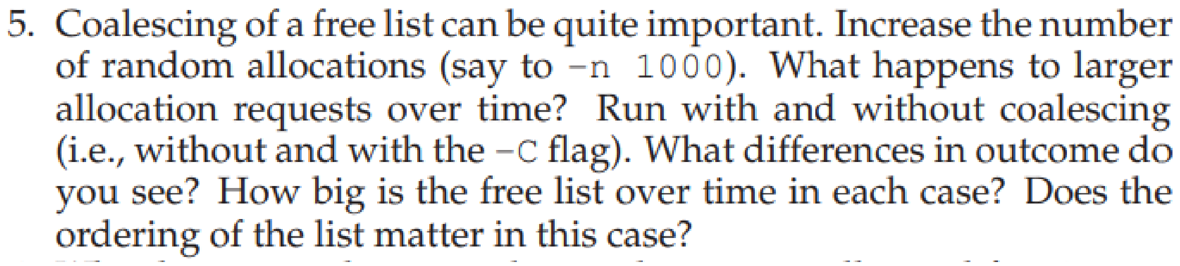


4. For the above questions, how the list is kept ordered can affect the time it takes to find a free location for some of the policies. Use the different free list orderings (`-l ADDRSORT`, `-l SIZESORT+`, `-l SIZESORT-`) to see how the policies and the list orderings interact.

- ADDRSORT has almost no difference on different policieds
- SIZESORT+ benefits the FIRST policy

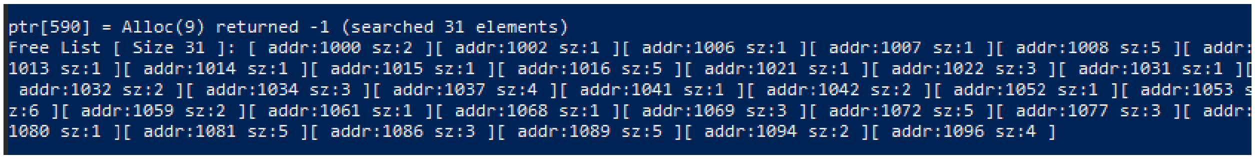
- SIZESORT- benefits the WORST policy

Question 5



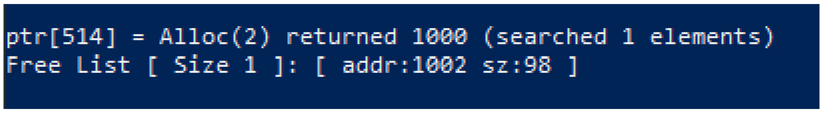
5. Coalescing of a free list can be quite important. Increase the number of random allocations (say to `-n 1000`). What happens to larger allocation requests over time? Run with and without coalescing (i.e., without and with the `-C` flag). What differences in outcome do you see? How big is the free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

The free list gets chopped up into small pieces if we do not coalesce the free space.



```
ptr[590] = Alloc(9) returned -1 (searched 31 elements)
Free List [ Size 31 ]: [ addr:1000 sz:2 ][ addr:1002 sz:1 ][ addr:1006 sz:1 ][ addr:1007 sz:1 ][ addr:1008 sz:5 ][ addr:
1013 sz:1 ][ addr:1014 sz:1 ][ addr:1015 sz:1 ][ addr:1016 sz:5 ][ addr:1021 sz:1 ][ addr:1022 sz:3 ][ addr:1031 sz:1 ][
 addr:1032 sz:2 ][ addr:1034 sz:3 ][ addr:1037 sz:4 ][ addr:1041 sz:1 ][ addr:1042 sz:2 ][ addr:1052 sz:1 ][ addr:1053 s
z:6 ][ addr:1059 sz:2 ][ addr:1061 sz:1 ][ addr:1068 sz:1 ][ addr:1069 sz:3 ][ addr:1072 sz:5 ][ addr:1077 sz:3 ][ addr:
1080 sz:1 ][ addr:1081 sz:5 ][ addr:1086 sz:3 ][ addr:1089 sz:5 ][ addr:1094 sz:2 ][ addr:1096 sz:4 ]
```

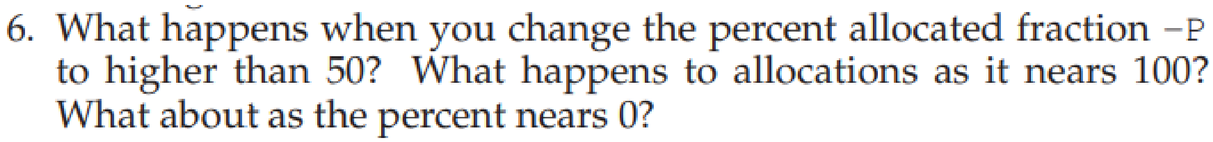
If we coalesce, large chunk of memory will be kept in the list.



```
ptr[514] = Alloc(2) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1002 sz:98 ]
```

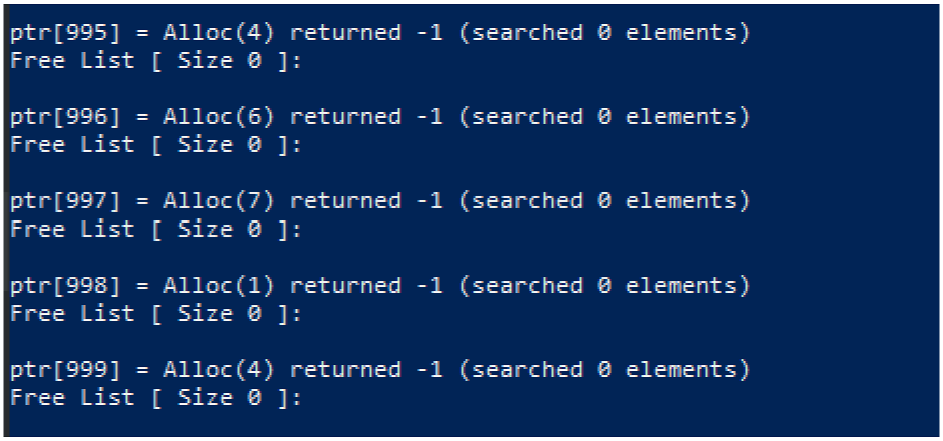
In this case, sort by address is better. We want to coalesce nearby address.

Question 6



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?

If the percent goes higher than 50, the memory will eventually run out and -1 will be returned.



```
ptr[995] = Alloc(4) returned -1 (searched 0 elements)
Free List [ Size 0 ]:

ptr[996] = Alloc(6) returned -1 (searched 0 elements)
Free List [ Size 0 ]:

ptr[997] = Alloc(7) returned -1 (searched 0 elements)
Free List [ Size 0 ]:

ptr[998] = Alloc(1) returned -1 (searched 0 elements)
Free List [ Size 0 ]:

ptr[999] = Alloc(4) returned -1 (searched 0 elements)
Free List [ Size 0 ]:
```