Freespace mgmt.

./malloc.py -S 100 -b 1000 -H 4 -a 4 -l ADDRSORT -p BEST -n 5

S => heap size, b => starting address
H => bytes of header per allocated block
a => allocated space rounds up to

ptr[0] = Alloc(3) returned?
list?

=> returned 1004 → start address (post the header) of allocated memory

since 1000 - 1004 is header
=> free list [size 1]: [addr:1008 sz:92]

Free(ptr[0]) → means free operation was successful
=> returned 0
=> Free list [size 2:] [addr:1000 sz:8
addr:1008 sz:92]

ptr[1] = Alloc(5)
⇒ returned 1012
⇒ Free list [size 2] [addr: 1000 sz: 8,
addr: 1020 sz 80]

Free(ptr[1])
⇒ returned 0
⇒ Free list [size 3] [addr: 1000 sz 8
addr: 1008 sz: 12,
addr: 1020 sz 80]

ptr[2] = Alloc(8)
⇒ returned 1012
⇒ Free list [size 2] [addr: 1000 sz 8
addr: 1020 sz 80]

(1) -n 10 -H 0 -p BEST -s 0
baseaddr = 1000

ptr[0] = Alloc(3)
returned 1000
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
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
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:8]

ptr[3] = Alloc(8)
returned 1008
free list [size:3], [addr:1000, sz:3],
[addr:1003, sz:5],
[addr:1016, sz:8]

Free(ptr[3])
returned 0
free list [size:4], [addr:1000, sz:3],
[addr:1003, sz:5],
[addr:1008, sz:8]
[addr:1016, sz:8]

ptr[4] = Alloc(2)
returned 1000

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
free list [size:4), [addr:1002, sz:1],
[addr: 1003, sz:5],
[addr: 1015, sz:1],
[addr: 1016, sz:84]

Free list grows over time, since coalesce is false.

(2) With policy "WORST"

Steps before ptr[3] = Alloc(8)
all same.

ptr[3] = Alloc(8)
returned 1016

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
free list = [size 5] : [⋮]
[addr: 1026, sz: 74]

ptr[5] = Alloc (7)
returned 1026

free list = [size 5]: [addr: 1000, sz: 3]
[addr: 1003, sz: 5]
[addr: 1008, sz: 9]
[addr: 10K, sz: 8]
[addr: 1033, sz: 67]

Free list is bigger with smaller sizes which weren't used because of "WORST" policy.

(3) With "FIRST" fit policy

Same result as with "BEST" for given problem.
But incase of "FIRST" the search exits quicker because it doesn't look at all the elements necessarily, finds the one that can fit and exits
So allocation can be faster.

(4) Try different free list orderings default above ⇒ addr sort

sizesort+ with BEST

⇒ Same, except order of free list for last step

ptr(5) = Alloc(7)
returned 1008
free list [size:4] [addr:1002, sz:1]
[addr:1015, sz:1]
[addr:1003, sz:5]
[addr:1016, sz:84]

× ↗ free list sorting
Actually this doesn't happen post allocation instead it happens on free operation (deallocation) or memory coalescing.

So same as before free list in this case

sizesort- with BEST

⇒ Reverse of free list, no other change. Since BEST fit will still search all elements so no difference there.

insert-front, insert-back with BEST

⇒ Again no change in terms of finding free locations. Just free-list order

is different

sizesort+/sizesort-, insertfront/back with WORST
⇒ same, since like BEST, WORST searches the whole free list only different is again ordering.

sizesort- with FIRST
⇒ will act like "WORST" since it will take bigger chunks

sizesort+ with FIRST
⇒ will act like "BEST" since it will take the smallest that fits

insertfront with FIRST
⇒ ptr[0] = Alloc(3)
returned 1000
freelist [size: 1] : [addr: 1003, sz: 97]

Free(ptr[0])
returned 0

freelist [size:2]: [addr:1000, sz:3],
[addr:1003, sz:97]

ptr[1] = Alloc(5)
returned 1003
freelist [size:2]: [addr:1008, sz:92]
[addr:1000, sz:3]

Reminder: free list ordering triggers on free not on allocation

Free(ptr[1])
returned 0
freelist [size:3]: [addr:1003, sz:5]
[addr:1008, sz:92]
[addr:1000, sz:3]

ptr[2] = Alloc(8)
returned 1008
freelist [size:3]: [addr:1016, sz:84]
[addr:1003, sz:5]
[addr:1000, sz:3]

Free(ptr[2])
returned 0

freelist [size:4] : [addr: 1008, sz:8]
→ [addr: 1016, sz:84]
→ [addr: 1003, sz:5]
→ [addr: 1000, sz:3]

ptr[3] = Alloc(8)
returned 1008
freelist [size: 3]: [addr: 1016, sz:84]
→ [addr: 1003, sz:5]
→ [addr: 1000, sz:3]

Free (ptr[3])
returned 0
freelist [size:4]: [addr: 1008, sz:8]
→ [addr: 1016, sz:84]
→ [addr: 1003, sz:5]
→ [addr: 1000, sz:3]

ptr(4) = Alloc (2)
returned 1008
freelist [size: 4]: [addr: 1010, sz:6]
⋮

ptr[5] = Alloc(7)
returned 1016
freelist [size:4]: [addr: 1010, sz: 6]
[addr: 1003, sz: 5]
[addr: 1000, sz: 3]
[addr: 1023, sz: 77]

insert-back with FIRST

ptr[0] = Alloc(3)
returned 1000
freelist: [size: 1]: [addr: 1003, sz: 97]

Free(ptr[6])
returned 0
freelist: [size: 2]: [addr: 1003, sz: 97]
[addr: 1000, sz: 3]

ptr[1] = Alloc(5)
returned 1003
freelist: [size: 2]: [addr: 1008, sz: 92]

[addr: 1000, sz: 3]

Free(ptr[1])
returned 0
freelist: [size: 3] : [addr: 1008, sz: 92]
[addr: 1000, sz: 3]
[addr: 1003, sz: 5]

ptr[2] = Alloc(8)
returned 1008
freelist: [size: 3] : [addr: 1016, sz: 84]
[addr: 1000, sz: 3]
[addr: 1003, sz: 5]

Free(ptr[2])
returned 0
freelist: [size: 4] : [addr: 1016, sz: 84]
⋮
[addr: 1008, sz: 8]

ptr[3] = Alloc(8)

returned 1016
freelist: [size:4] : [addr: 1024, sz:76]

Free (ptr[3])
returned 0
freelist: [size:5] : [addr:1016, sz:8]

Ptr[4] = Alloc(2)
returned 1024
freelist: [size:5] ; [addr:1026, sz:74]

Ptr[5] = Alloc(7)
returned 1026
freelist: [size:5] : [addr:1033, sz: 67]

This is similar to result for "WORST" since above also ends up picking the biggest chunk due to being FIRST

(5) -n 100 -H 0 -p BEST -s 0 -c

```
Free(ptr[53])
returned 0
Free List [ Size 20 ]: [ addr:1000 sz:2 ][ addr:1002 sz:1 ][ addr:1006 sz:1 ]
[ addr:1007 sz:1 ][ addr:1013 sz:1 ][ addr:1014 sz:1 ][ addr:1015 sz:1 ][ add
r:1021 sz:1 ][ addr:1022 sz:3 ][ addr:1031 sz:1 ][ addr:1032 sz:2 ][ addr:103
4 sz:3 ][ addr:1037 sz:4 ][ addr:1041 sz:1 ][ addr:1042 sz:2 ][ addr:1048 sz:
4 ][ addr:1052 sz:1 ][ addr:1069 sz:3 ][ addr:1081 sz:8 ][ addr:1096 sz:4 ]
```

BEST with coalesce

```
Free(ptr[53])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:7 ][ addr:1019 sz:2 ][ addr:1046 sz:1 ][
 addr:1071 sz:29 ]
```

SIZESORT- with coalesce

```
Free(ptr[53])
returned 0
Free List [ Size 17 ]: [ addr:1089 sz:11 ][ addr:1048 sz:4 ][ addr:1044 sz:4
][ addr:1069 sz:3 ][ addr:1023 sz:2 ][ addr:1014 sz:2 ][ addr:1000 sz:2 ][ ad
dr:1042 sz:2 ][ addr:1036 sz:1 ][ addr:1030 sz:1 ][ addr:1013 sz:1 ][ addr:10
06 sz:1 ][ addr:1052 sz:1 ][ addr:1041 sz:1 ][ addr:1007 sz:1 ][ addr:1002 sz
:1 ][ addr:1022 sz:1 ]
```

sizesort+ with coalesce

```
Free(ptr[53])
returned 0
Free List [ Size 15 ]: [ addr:1019 sz:1 ][ addr:1038 sz:1 ][ addr:1037 sz:1 ]
[ addr:1005 sz:1 ][ addr:1036 sz:1 ][ addr:1029 sz:1 ][ addr:1011 sz:2 ][ add
r:1034 sz:2 ][ addr:1006 sz:5 ][ addr:1050 sz:3 ][ addr:1016 sz:3 ][ addr:105
3 sz:3 ][ addr:1096 sz:4 ][ addr:1056 sz:4 ][ addr:1084 sz:7 ]
```

Yes ordering matters because coalesce happens only if elements are adjacent, Hence addrsort worksss

the best. ↓

(simulator code only checks first element with list)

In reality it depends how frequently coalesce is run, but in other sorts they are far away unlike addrsort when adjacent.

6 with -P 60 → percent allocated fraction allocs in generated ops

All BEST policy

```
Free(ptr[55])
returned 0
Free List [ Size 9 ]: [ addr:1017 sz:1 ][ addr:1020 sz:1 ][ addr:1021 sz:1 ][
 addr:1022 sz:6 ][ addr:1038 sz:1 ][ addr:1039 sz:1 ][ addr:1045 sz:5 ][ addr
:1066 sz:4 ][ addr:1090 sz:10 ]
```

with -P 60 with coalesce

```
Free(ptr[55])
returned 0
Free List [ Size 5 ]: [ addr:1009 sz:6 ][ addr:1026 sz:1 ][ addr:1047 sz:6 ][
 addr:1056 sz:1 ][ addr:1084 sz:16 ]
```

with -P 95

```
ptr[89] = Alloc(1) returned -1 (searched 0 elements)
Free List [ Size 0 ]:
```

with -P 3

```
Free(ptr[49])
returned 0
Free List [ Size 22 ]: [ addr:1000 sz:3 ][ addr:1003 sz:1 ][ addr:1004 sz:1 ]
[ addr:1005 sz:2 ][ addr:1007 sz:1 ][ addr:1008 sz:4 ][ addr:1012 sz:1 ][ add
r:1013 sz:2 ][ addr:1015 sz:5 ][ addr:1020 sz:1 ][ addr:1021 sz:1 ][ addr:102
2 sz:1 ][ addr:1023 sz:1 ][ addr:1024 sz:1 ][ addr:1025 sz:6 ][ addr:1031 sz:
1 ][ addr:1032 sz:1 ][ addr:1033 sz:1 ][ addr:1034 sz:9 ][ addr:1043 sz:1 ][
addr:1044 sz:10 ][ addr:1054 sz:46 ]
```

with -F 5 with coalesce

```
Free(ptr[49])
returned 0
Free List [ Size 1 ]: [ addr:1000 sz:100 ]
```

① Highly fragmented free space

baseaddr

-A +10+5+10+10.-1+10-2 1000 size = 100

1050

1080

1100

For highly fragmented WORST would work best, giving bigger free list

Also allocation of smaller sizes then freed would be left unused if allocation post that are bigger IDK sizes

```
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder SIZESORT-
coalesce False
numOps 10
range 10
percentAlloc 50
allocList +4,-0,+5,+10,+5,-2,+6,-4,+7,+8,-6,+25,+20,-7
compute True
```

addsort

With a... WORST

```
Free(ptr[7])
returned 0
Free List [ Size 6 ]: [ addr:1000 sz:4 ][ addr:1009 sz:10 ][ addr:1024 sz:6 ]
[ addr:1037 sz:8 ][ addr:1045 sz:25 ][ addr:1090 sz:10 ]
```

BEST

```
Free(ptr[7])
returned 0
Free List [ Size 6 ]: [ addr:1000 sz:4 ][ addr:1009 sz:6 ][ addr:1015 sz:4 ][
 addr:1031 sz:8 ][ addr:1039 sz:25 ][ addr:1084 sz:16 ]
```

FIRST

```
Free(ptr[7])
returned 0
Free List [ Size 6 ]: [ addr:1000 sz:4 ][ addr:1009 sz:6 ][ addr:1015 sz:4 ][
 addr:1031 sz:8 ][ addr:1039 sz:25 ][ addr:1084 sz:16 ]
```

With FIRST sizesort-

```
Free(ptr[7])
returned 0
Free List [ Size 6 ]: [ addr:1045 sz:25 ][ addr:1090 sz:10 ][ addr:1009 sz:10
 ][ addr:1037 sz:8 ][ addr:1024 sz:6 ][ addr:1000 sz:4 ]
```

With FIRST insert-back

```
Free(ptr[7])
returned 0
Free List [ Size 6 ]: [ addr:1090 sz:10 ][ addr:1000 sz:4 ][ addr:1009 sz:10
][ addr:1024 sz:6 ][ addr:1037 sz:8 ][ addr:1045 sz:25 ]
```