Mark-Sweep Collection

- Garbage detection, i.e., distinguishing live objects from garbage objects through some means.
  - Done by tracing: starting at the root set and actually traversing the graph of pointer relationships, usually by either a depth-first or a breadth-first traversal.
  - Reached objects are *marked*, either by altering bits within them, or by recording them in a bitmap or some other similar table structure.

Mark-Sweep Collection

- Garbage detection, i.e., distinguishing live objects from garbage objects through some means.
  - Done by tracing: starting at the root set and actually traversing the graph of pointer relationships, usually by either a depth-first or a breadth-first traversal.
  - Reached objects are *marked*, either by altering bits within them, or by recording them in a bitmap or some other similar table structure.
- Storage reclamation, i.e., reclaiming the storage of the garbage objects so that it is available for use by the mutator.
  - Memory is *swept*, i.e., exhaustively examined, to find all of the unmarked objects and reclaim their space by linking them onto one or more free lists.

Problems With Mark-Sweep Collection

- Difficult to handle objects of varying sizes without fragmentation of the available memory.
  - This is basically the external fragmentation problem.
  - Can be mitigated by having binned free lists.

Problems With Mark-Sweep Collection

- Difficult to handle objects of varying sizes without fragmentation of the available memory.
  - This is basically the external fragmentation problem.
  - Can be mitigated by having binned free lists.
- Cost of a collection is proportional to the size of the heap, including both live and garbage objects.
  - Similar to the issue with an implicit free list allocator.

Problems With Mark-Sweep Collection

- Difficult to handle objects of varying sizes without fragmentation of the available memory.
  - This is basically the external fragmentation problem.
  - Can be mitigated by having binned free lists.
- Cost of a collection is proportional to the size of the heap, including both live and garbage objects.
  - Similar to the issue with an implicit free list allocator.
- Can worsen locality of reference, especially at the page level.
  - The working set of the mutator can be scattered across too many virtual memory pages.
  - May be somewhat mitigated by empirically observed patterns of clustered object allocation and lifetimes.

Mark-Compact Collection

- A variant of mark-sweep that remedies the fragmentation and locality problems of mark-sweep collectors.
  - Mark phase is unchanged.
  - Live objects are then *compacted*, i.e., moved until all live objects are contiguous.
  - This compaction is usually done by a linear scan through memory, finding live objects and “sliding” them down to be adjacent to the previous live object.
  - This implicitly coalesces all the memory holes into a large contiguous free block at one end of the heap.

Mark-Compact Collection

- A variant of mark-sweep that remedies the fragmentation and locality problems of mark-sweep collectors.
  - Mark phase is unchanged.
  - Live objects are then *compacted*, i.e., moved until all live objects are contiguous.
  - This compaction is usually done by a linear scan through memory, finding live objects and "sliding" them down to be adjacent to the previous live object.
  - This implicitly coalesces all the memory holes into a large contiguous free block at one end of the heap.
- Cost-benefit tradeoff: Needs several passes over the heap to compact.
  - Sliding compactors make two or three passes over the live objects.
  - One pass computes the new locations for the objects.
  - Subsequent passes update pointers to refer to these new locations, and to actually move the objects.
  - May be slower than mark-sweep if the fraction of live data is large.