CSE 350: Advanced Data Structures and Indexes (Spring 2026)

Lecture 3: External Memory Model
Common Techniques & Cost Analysis

1/29/2026

University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

Example: from list to heap file

- Example: a collection of unordered items -- assuming everything is unique
  - Need to support
    - insert(x)
    - delete(x)
    - lookup(x): reports whether the item x is found
  - In-memory solution:

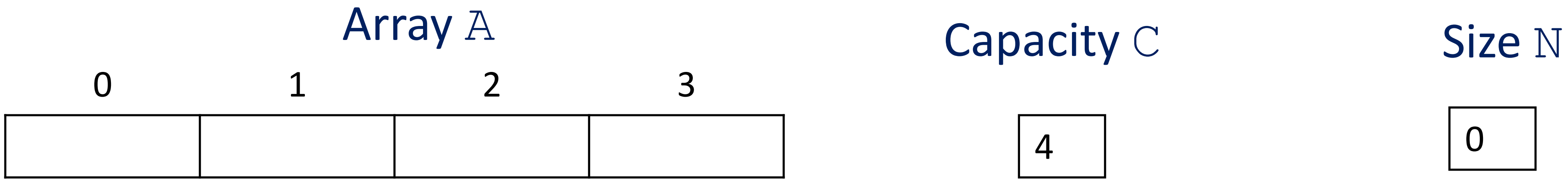
Array

size
4

0	1	2	3	4	5	6
100	200	300	400			

Algorithms of list

```
// List L: (A, C, N)
```




```
def init():
  return (array(4), 4, 0)

def insert(L, x):
  if L.N >= L.C then
    resize(L, L.C * 2)
  A[L.N] <- x
  L.N <- L.N + 1
```

```
def resize(L, C2):
  A2 <- array(C2)
  N2 <- min(L.N, C2)
  A2[0..N2-1] <- A[0..N2-1]
  L <- (A2, C2, N2)
```

Complexity analysis of list

Assumptions:

In-memory: count # of steps

What counts as a step:

- assignment of each word <-
- arithmetic operations (+,-,*,/,%,…)
- allocation/deallocation of memory -- regardless of size

What about branch & loop? Depends

- Worst-case analysis
  - Count the maximum number of steps
- Average-case analysis
  - average number of steps
    may need to derive/assume distribution

complexity for size N	In-memory List
insert(x)	
delete(x)	
lookup(x)	

```
def insert(L, x):
  if L.N >= L.C then
    resize(L, L.C * 2)
  A[L.N] <- x
  L.N <- L.N + 1

def resize(L, C2):
  A2 <- array(C2)
  N2 <- min(L.N, C2)
  A2[0..N-1] <- A[0..N-1]
  L <- (A2, C2, N2)
```

Asymptotic analysis

- Only trend matters, how to formalize this?
  - Mathematical tool: limit

$f: N \to R$

$$\lim_{n \to +\infty} f(n) = L \quad iff$$

$$\forall \varepsilon > 0, \exists n_0 \in N, \forall n \geq n_0, |f(n) - L| \leq \varepsilon$$

Asymptotic analysis

- Only trend matters, how to formalize this?
  - Mathematical tool: limit

$f: N \to R$

$$\lim_{n \to +\infty} f(n) = L \quad iff$$

$$\forall \varepsilon > 0, \exists n_0 \in N, \forall n \geq n_0, |f(n) - L| \leq \varepsilon$$

- How to use it to express trends? Intuition:
  - constants c_0 and c_1 has the same trend, regardless of how much they differ
  - polynomials of the same degree have the same trend, regardless of their parameters
  - polynomials of a higher degree has a faster trend to increase in cost

Suppose $f_1(n) = c_0, f_2(n) = c_1, f_3(n) = 3n + 2, f_4(n) = 5n - 4, f_5(n) = n^2 + n - 3$

Want to define $(" \prec "," \equiv ")$ such that $f_1 \equiv f_2 \prec f_3 \equiv f_4 \prec f_5$

Does this work: two function have the same trend if their difference is bounded by constant?

No! Counter example: $\lim_{n \to +\infty} |f_3(n) - f_4(n)| = \lim_{n \to +\infty} |2n - 6|$ which does not exist

Asymptotic analysis

- Only trend matters, how to formalize this?
  - Mathematical tool: limit

$$f: N \to R^+ \qquad \lim_{n\to+\infty} f(n) = L \;\; iff$$

$$\forall \varepsilon > 0, \exists n_0 \in N, \forall n \geq n_0, |f(n) - L| \leq \varepsilon$$

- How to use it to express trends? Intuition:
  - constants c_0 and c_1 has the same trend, regardless of how much they differ
  - polynomials of the same degree have the same trend, regardless of their parameters
  - polynomials of a higher degree has a faster trend to increase in cost

Suppose $f_1(n) = c_0, f_2(n) = c_1, f_3(n) = 3n + 2, f_4(n) = 5n - 4, f_5(n) = n^2 + n - 3$

Want to define (" $\prec$ "," $\equiv$ ") such that $f_1 \equiv f_2 \prec f_3 \equiv f_4 \prec f_5$

Does this work: two function have the same trend if their difference is bounded by constant?

No! Counter example: $\lim_{n\to+\infty} |f_3(n) - f_4(n)| = \lim_{n\to+\infty} |2n - 6|$ which does not exist

Asymptotic analysis

- Whether the limit of the ratio of two functions $\lim_{n \to +\infty} \frac{f(n)}{g(n)}$
  - does not exist (i.e., tends to $+\infty$) => $f(n)$ grows faster than $g(n)$
  - is a non-zero => same trend
  - is zero => $f(n)$ grows slower than $g(n)$

Suppose $f_1(n) = c_0, f_2(n) = c_1, f_3(n) = 3n + 2, f_4(n) = 5n - 4, f_5(n) = n^2 + n - 3$

Want to define $(\prec, \equiv)$ such that $f_1 \equiv f_2 \prec f_3 \equiv f_4 \equiv f_5$

$$\lim_{n \to +\infty} \frac{f_1(n)}{f_2(n)} = \lim_{n \to +\infty} \frac{c_0}{c_1} = \frac{c_0}{c_1} \qquad \Rightarrow f_1 \text{ and } f_2 \text{ has the same trend}$$

$$\lim_{n \to +\infty} \frac{f_1(n)}{f_3(n)} = \lim_{n \to +\infty} \frac{c_0}{3n+2} = 0 \qquad \Rightarrow f_1 \text{ grows slower than } f_3$$

$$\lim_{n \to +\infty} \frac{f_5(n)}{f_3(n)} = \lim_{n \to +\infty} \frac{n^2+n-3}{3n+2} \to +\infty \qquad \Rightarrow f_5 \text{ grows faster than } f_3$$

Asymptotic analysis

- Formally, we define following sets of all functions $g: N \rightarrow R^+$ for $f: N \rightarrow R^+$
  - $o(f) = \left\{ g \in {R^+}^N \middle| \lim_{n \to +\infty} \frac{g(n)}{f(n)} = 0 \right\}$
  - $\omega(f) = \left\{ g \in {R^+}^N \middle| \lim_{n \to +\infty} \frac{f(n)}{g(n)} = 0 \right\}$
  - $\Theta(f) = \left\{ g \in {R^+}^N \middle| \exists c \in R^+, \lim_{n \to +\infty} \frac{f(n)}{g(n)} = c \right\}$
  - Big oh and Big Oemga:
    - $O(f) = \Theta(f) \cup o(f)$
    - $\Omega(\mathrm{f}) = \Theta(f) \cup \omega(f)$

Examples:

$$2n^2 - 4 \in \mathrm{O}(\mathrm{n}^2)$$

means

$2n^2 - 4$ grows at most as fast as n^2

For convenience, we often write "=" instead of "∈"

i.e., $2n^2 - 4 = O(n^2)$

Complexity analysis of list

Assumptions:

In-memory: count # of steps

What counts as a step:

- assignment of each word <-
- arithmetic operations (+,-,*,/,%,…)
- allocation/deallocation of memory -- regardless of size

What about branch & loop? Depends

- Worst-case analysis
  - Count the maximum number of steps
- Average-case analysis
  - average number of steps
    may need to derive/assume distribution

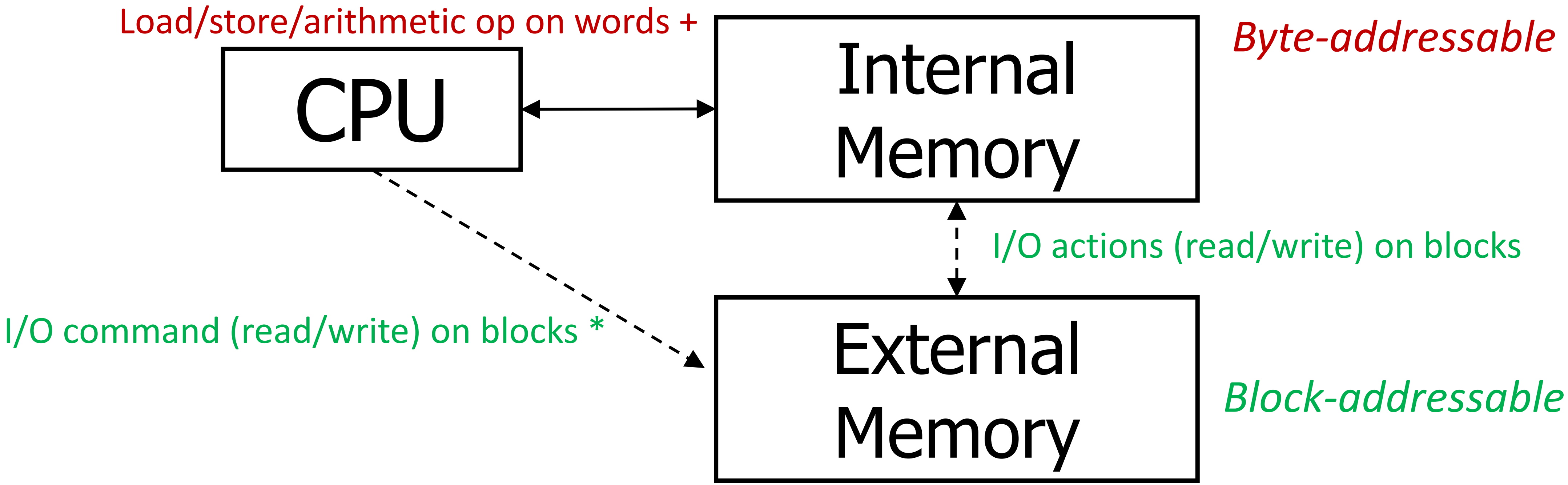
```
def insert(L, x):
  if L.N >= L.C then
    resize(L, L.C * 2)
  A[L.N] <- x
  L.N <- L.N + 1

def resize(L, C2):
  A2 <- array(C2)
  N2 <- min(L.N, C2)
  A2[0..N-1] <- A[0..N-1]
  L <- (A2, C2, N2)
```

complexity for size N	In-memory List
insert(x)	$\Theta(N)$ (worst case) $\Theta(1)$ (average case)
delete(x)	
lookup(x)	

External Memory (EM) Model

- Two levels in storage hierarchy
  - I/O latency dominates computation/memory access latencies
  - Complexity analysis will focus on # of I/Os (i.e., # of blocks read/written)



* One block is a fixed number of consecutive bytes (e.g., 512 B, 4 KB), aligned to modulo = 0 boundaries.
+ A word is a unit of consecutive bytes for operations (e.g., a 4-byte integer).

Complexity Analysis for EM model

Assumptions:

EM model: count # of I/O operations

What counts as an I/O operation?

- reading up to a page
- writing up to a page
- If a read/write crosses page boundaries, count both

```
Assuming 4 KiB pages
ssize_t pread(int fd, void *buf, size_t count, off_t offset);
ssize_t pwrite(int fd, const void *buf, size_t count, off_t offset);
```

Operation	# I/O
`pread(fd, buf, 4096, 0)`	1
`pwrite(fd, buf, 4096, 0)`	1
`pread(fd, buf, 4096, 10 * 4096)`	1
`pread(fd, buf, 8, 512)`	1
`pread(fd, buf, 4096, 2048)`	2

Example: from list to heap file

- Example: a collection of unordered items -- assuming everything is unique
  - Need to support
    - insert(x)
    - delete(x)
    - lookup(x): reports whether the item x is found
  - In-memory solution:

size

4

Array

0	1	2	3	4	5	6
100	200	300	400			

External memory solution?

How about replace all memory loads with 4-byte disk read, and all memory stores with 4-byte disk writes?

Random access is expensive

How about replace all memory loads with 4-byte disk read, and all memory stores with 4-byte disk writes?

```
def insert(L, x):
  if L.N >= L.C then
    resize(L, L.C * 2)
  A[L.N] <- x
  L.N <- L.N + 1

def resize(L, C2):
  A2 <- array(C2)
  N2 <- min(L.N, C2)
  A2[0..N-1] <- A[0..N-1]
  L <- (A2, C2, N2)
```

```
L is a file descriptor to a file

Bytes 0 - 3: N
Bytes 4(i + 1) - 4(i + 1) + 3: ith value

def insert_heap(L, x):
  pread(L, &N, 4, 0)
  N <- N + 1
  pwrite(L, &N, 4, 0)
  pwrite(L, &x, 4, 4 * N)
```

complexity for size N	In-memory List	Naïve heap file
insert(x)	$\Theta(N)$ (worst case) $\Theta(1)$ (average case)	5 = $\Theta(1)$ (worst case)
delete(x)		
lookup(x)		

But, 5 I/O is around
50 ms for magnetic disks
or 500 μs for SATA NAND SSD

in-memory: <= hundreds of ns

Caching and page-granular I/O

```
L is a file descriptor to a file

Bytes 0 - 3: N
Bytes 4(i + 1) - 4(i + 1) + 3: ith value

def insert_heap(L, x):
  pread(L, &N, 4, 0)
  N <- N + 1
  pwrite(L, &N, 4, 0)
  pwrite(L, &x, 4, 4 * N)
```

```
def init_heap_with_cache(fd):
  pread(L, p, 4096, 0)
  pn <- 0
  N <- p[0:4]
  return (fd, N, pn, p) // L
def insert_heap_with_cache(L, x):
  pn2 <- (L.N + 1) / 4096
  off <- (L.N + 1) % 4096
  if pn2 != L.pn then
    pwrite(L.fd, L.p, 4096, L.pn * 4096)
    pread(L.fd, L.p, 4096, pn2 * 4096)
    L.pn = pn2
  L.p[off:off + 4] <- x
```

complexity for size N	In-memory List	Naïve heap file
insert(x)	$\Theta(N)$ (worst case) $\Theta(1)$ (average case)	5 = $\Theta(1)$ (worst case)
delete(x)		
lookup(x)		

```
def close_heap_with_cache(L):
  // write cached page L.p
  // read page 0
  // update N
  // write page 0
```

Caching and page-granular I/O

```
L is a file descriptor to a file

Bytes 0 - 3: N
Bytes 4(i + 1) - 4(i + 1) + 3: ith value

def insert_heap(L, x):
  pread(L, &N, 4, 0)
  N <- N + 1
  pwrite(L, &N, 4, 0)
  pwrite(L, &x, 4, 4 * N)
```

→

```
def init_heap_with_cache(fd):
  pread(L, p, 4096, 0)
  pn <- 0
  N <- p[0:4]
  return (fd, N, pn, p) // L
def insert_heap_with_cache(L, x):
  pn2 <- (L.N + 1) / 4096
  off <- (L.N + 1) % 4096
  if pn2 != L.pn then
    pwrite(L.fd, L.p, 4096, L.pn * 4096)
    pread(L.fd, L.p, 4096, pn2 * 4096)
    L.pn = pn2
  L.p[off:off + 4] <- x
```

complexity for size N	In-memory List	Naïve heap file	Heap file with one page cache
insert(x)	$\Theta(N)$ (worst case) $\Theta(1)$ (average case)	5 = $\Theta(1)$ (worst case)	2 = $\Theta(1)$ (worst case) 2/1024 = $\Theta(1)$ (average case)
delete(x)			
lookup(x)			

Additional problems?

- Concurrency?
- Crash & recovery?
- How to further improve performance?

```
def init_heap_with_cache(fd):
  pread(L, p, 4096, 0)
  pn <- 0
  N <- p[0:4]
  return (fd, N, pn, p) // L
def insert_heap_with_cache(L, x):
  pn2 <- (L.N + 1) / 4096
  off <- (L.N + 1) % 4096
  if pn2 != L.pn then
    pwrite(L.fd, L.p, 4096, L.pn * 4096)
    pread(L.fd, L.p, 4096, pn2 * 4096)
    L.pn = pn2
  L.p[off:off + 4] <- x

def close_heap_with_cache(L):
  // write cached page L.p
  // read page 0
  // update N
  // write page 0
```